НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

Дипломна робота

на тему: Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища

Студент групи ТВ-51 Нестерко Євген Павлович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник роботи ст. в., Ляшенко Максим Володимирович _____
(вчені ступінь та звання, прізвище, ініціали) (підпис)

Кількість сторінок 64

Кількість ілюстрацій 15

Київ – 2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища

Виконав: студент 4 курсу, групи ТВ-51

Нестерко Євген Павлович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач, Ляшенко Максим Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент, Баранюк Олександр Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2018 р.

ЗАВДАННЯ

на дипломну роботу студенту

Нестерко Євгену Павловичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища”

керівник роботи старший викладач, Ляшенко Максим Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Microsoft Windows, мова програмування Java, браузер та Інтернет.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення щодо організації хмарного сховища та авторизації, обґрунтувати обрані програмні рішення та технології, розробити програмне забезпечення, що виконує дані завдання та зробити висновки за результатами роботи.
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень) Актуальність, Мета та завдання, Основні модулі функціоналу системи, Опис функціональності , Архітектура авторизації, Технології та програмні засоби,

Підключення та конфігурація OAauth2, Ідентифікатор клієнта для типу "Веб-додаток", Інтерфейс модулю авторизації за допомогою OAuth2, Інтерфейс контролю користувачів адміністратора, Інтерфейс надання доступу до файлу іншому користувачеві, Таблиця у базі даних для зберігання співвідношення “Файл - Користувач -Доступ”, Висновки.

Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	20.03.19 - 04.04.19	
2.	Розробка архітектури та загальної структури системи	04.04.19 - 15.04.19	
3.	Розробка структур окремих підсистем	15.04.19 - 25.04.19	
4.	Підготовка матеріалів	25.04.19 - 06.05.19	
5.	Програмна реалізація системи	06.05.19 - 16.05.19	
6.	Захист програмного продукту	17.05.19	
7.	Оформлення пояснювальної записки	17.05.19 - 30.05.19	
8.	Передзахист	31.05.19	
9.	Захист		

Студент _____ Нестерко Є.П
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ляшенко М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи було створення системи для зберігання даних, спільної роботи з даними у хмарному сховищі та організація контрольованого доступу до даних, що знаходяться у хмарному сховищі. Для організації контрольованого доступу було впроваджено засоби аутентифікації до системи, та засоби, що контролюють доступ користувачів до файлів у системі. За допомогою керуючого модуля системи, можна надавати певним користувачам доступ до файлу. Адміністратор системи може додавати та видаляти користувачів із системи і також переглядати доступні йому файли, що знаходяться у сховищі. Користувачі можуть об’єднуватись у групи та переглядати файли, що доступні цій групі. Користувач може вийти з групи, створити свою власну та запросити користувачів до своєї групи. Усі дані про користувачів, їх рівні доступу та дані про файли, що знаходяться у хмарному сховищі, записуються у базу даних. Ключові слова: Spring, Хмарне Сховище, OAuth2, Heroku, PostgreSQL. Записка містить 64 сторінки, 15 рисунків, 3 додатки і 22 посилання.

ABSTRACT

The purpose of the work was to create a system for storing data, collaborating with data in a cloud storage and organizing controlled access to data in a cloud storage. For the organization of controlled access, the means of authentication to the system were implemented, and the means controlling the access of users to files in the system. With the system control module, you can grant certain users access to the file. System administrators can add and remove users from the system, and also view available files that are in the repository. Users can join groups and view files that are available to this group. In turn, the user can leave the group, create their own and invite users to their group. All data about users, their access levels, and data about files in the cloud storage are recorded in the database. Keywords: Spring, Cloud Storage, OAuth2, Heroku, PostgreSQL. The note contains 64 pages, 15 figures, 3 annexes and 22 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

MVC – (англ. model-view-controller) – це архітектурний зразок, який зазвичай використовується для розробки інтерфейсів користувача, що розділяє додаток на три взаємопов'язані частини;

API (англ. Application Programming Interface) – являє собою набір визначень підпрограм, комунікаційних протоколів і інструментів для побудови програмного забезпечення;

CS (англ. Cloud Storage) – є моделлю комп'ютерного зберігання даних, в якій цифрові дані зберігаються на віддалених серверах;

СУБД – система управління базами даних;

UI (англ. user interface) – в галузі промислового дизайну взаємодії людина-комп'ютер – це простір, де відбуваються взаємодії між людьми і машинами;

AD (англ. Active Directory) – це хмарна служба управління посвідченнями і доступом від корпорації Майкрософт;

IoT (англ. Internet of Things) – концепція мережі, що складається із взаємозв'язаних фізичних пристроїв, які мають вбудовані давачі, а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між фізичним світом і комп'ютерними системами, за допомогою використання стандартних протоколів зв'язку;

AI (англ. Artificial intelligence) – розділ комп'ютерної лінгвістики та інформатики, що опікується формалізацією проблем та завдань, які подібні до дій, які виконує людина.

IaaS (англ. Infrastructure-as-a-Service) – инфраструктура у ролі сервісу.

PaaS(англ. Platform-as-a-Service) – платформа у ролі сервісу.

SaaS(англ. Software-as-a-Service) – програмне забезпечення у ролі сервісу.

IDE (англ. integrated development environment) – це програма, яка надає комплексні можливості програмістам для розробки програмного забезпечення.

ВСТУП

Сучасний стрімкий розвиток інформаційних технологій потребує нового погляду на зберігання даних та їх безпеку. Саме тому на даний момент більшої популярності набуває використання хмарних сховищ.

На відміну від традиційних пристроїв зберігання даних, таких як жорсткі диски або компакт-диски, де інформація зберігається у фізичному форматі, всі дані в хмарі зберігаються практично на серверах. Крім того, можна отримати доступ до всього, що знаходиться на хмарі, з будь-якого комп'ютера, що має підключення до Інтернету.

Першою основною відмінністю між хмарою та традиційними засобами зберігання є доступність. Хоча жорсткі диски пропонують локальний доступ до інформації, хмарне сховище надає необмежений доступ до даних, якщо користувачі можуть надати необхідні ключі доступу і якщо комп'ютер має доступ до Інтернету.

Робота з сервісом хмарних сховищ має багато переваг. Напевно, найочевиднішою перевагою, є те, що вам не доведеться загороджувати жорсткий диск марною інформацією. Крім того, служби хмарного зберігання зазвичай використовують передові функції стиснення інформації, що означає більшу можливість індексування.

Це, у свою чергу, означає велику швидкість пошуку. Однак використання служби хмарних сховищ не позбавлено недоліків. Будучи хостингом, хмара значною мірою залежить від швидкості підключення до Інтернету.

Необхідною деталью у хмарному сховищі є безпека. Самое тому постає питання щодо організації доступу до хмарного сховища, а також доступності до файлів у самому сховищі.

Пропонується систему, за допомогою якої можна буде користуватись хмарним сховищем та організувати контрольований доступ до даних у сховищі.

Програма представляє собою веб-застосунок, який надає сервіс зберігання даних у хмарному середовищі з контрольованим доступом до даних.

Для організації контрольованого доступу до самого сховища пропонується використовувати технологію OAuth2, а для контролю доступу до файлів у сховищі – зберігати інформацію про рівні доступу у базі даних.

Записка містить 5 розділів.

У першому розділі аналізуються проблеми зберігання даних.

У другому розділі проводиться огляд існуючих рішень щодо зберігання даних у файлових сховищах.

У третьому розділі розглядаються засоби програмної реалізації.

У четвертому розділі описується програмна реалізація системи хмарного файлового сховища та контрольований доступ до неї.

У п’ятому розділі описується методика роботи із системою.

1 АНАЛІЗ ТА ВИРІШЕННЯ ПРОБЛЕМИ ЗБЕРІГАННЯ ДАНИХ

У сучасному світі з великими обсягами даних значною мірою фокус підприємств спрямовується на аналітиці; іншими словами, центральною проблемою стає те, що робити з усіма даними та файлами, які маються. Це важлива проблема для вирішення, але неможливо це вирішити, якщо немає ефективного, довгострокового рішення для зберігання даних, щоб забезпечити стабільну основу. Врешті-решт, неможливо оперувати даними, які не маєте куди діти.

1.1 Список проблем зберігання даних

Тому маємо список проблем, які необхідно розглянути:

1. Вартість. Запуск власного центру зберігання даних є дорогою операцією. Необхідно витратити гроші на початкове налаштування, поточне обслуговування та витрати, пов'язані з людьми, відповідальними за його підтримку. Знову ж таки, найкращим рішенням тут є аутсорсинг роботи; доведеться заплатити щомісячну плату, але це заощадить гроші в довгостроковій перспективі.

2. Пошкодження. Практично кожна форма зберігання даних має потенціал для пошкодження. Усілякі фізичні частинки можуть перешкоджати більшості форм зберігання даних, і все, що покладається на магнітні смуги або електричні накопичувачі, може бути пошкоджено електромагнітними перешкодами. Навіть якщо немає зовнішнього джерела, що безпосередньо втручається в нього, дані можуть з часом природним чином погіршитися. Найкраще для захисту тут використовується декілька резервних копій.

3. Безпека. Безпека є головною проблемою, яку потрібно подолати. Гіпотетично, якщо дані зберігаються “де-небудь” і “як-небудь”, третій стороні дуже легко буде отримати доступ до даних. Існує багато рівнів безпеки, які можуть допомогти запобігти цьому несанкціонованому доступу, включаючи шифрування та залежність від сторонніх постачальників, але існує обмеження щодо того, наскільки вони можуть захистити. Необхідно виконати жорсткі операції, вибрати найкращих партнерів і постійно дотримуватися найкращих практик команди.

4. Інфраструктура. Дані потребують місця для зберігання, так само, як об'єкти потребують полиці або контейнера; дані повинні займати деякий простір. Якщо планується зберігати величезну кількість даних, то знадобиться інфраструктура, необхідна для її зберігання, що часто означає інвестування в високотехнологічні сервери, які займуть значний простір в офісі або будівлі.

5. Сумісність. Якщо планується використовувати декілька систем або програм зі своїми даними, потрібно переконатися, що вони сумісні. Для цього потрібно знайти партнера зі зберігання даних з відкритим API і чисту систему переходу.

6. Масштаб. Можливо, можна знайти рішення для зберігання даних, яке відповідає поточним потребам належним чином, але що станеться, якщо ці потреби раптово зміниться? Іноді потрібно подивитись вперед хоча б на 5 років. Дане рішення для зберігання даних потребує певних можливостей для масштабування. Тут варто продумати якомога більше варіантів, оскільки немає впевненості, як саме зміняться потреби у майбутньому.

7. Інтерфейс користувача та доступність. Дані не будуть корисні, якщо до них важко отримати доступ; зрештою, зберігання даних - це лише тимчасовий захід, щоб була можливість згодом оперувати даними та використовувати їх. Відповідно, знадобляться системи з інтуїтивним, доступним інтерфейсом користувача (UI) і чистою доступністю для будь-якої потрібної функціональності [19].

Проаналізувавши дані проблеми, можна зробити висновок, що одним з найпростіших способів вирішення проблем є використання хмарного хостингу та хмарного сховища, які використовують переваги інфраструктури іншої компанії, щоб заощадити цей простір і проблеми з самостійним налаштуванням.

Хмарні обчислення - це обчислювальна модель за вимогою, яка спирається на стабільне підключення до Інтернету для обміну та доступу до даних на різних пристроях, таких як комп'ютери або смартфони.

Якщо обернутися назад, щоб побачити більш широку картину, хмарні обчислення дозволяють користувачам отримувати доступ до всієї інформації, що зберігається в сторонньому центрі обробки даних. Це досягається за допомогою мінімальних ресурсів і мінімального контролю.

Історично кажучи, хмарні обчислення, як функціональна комп'ютерна модель, не є новим. Насправді, так звана «хмара» йде з 70-х і 80-х років, коли ARPANET і CNET, попередники сучасного Інтернету, вже натякали на те, що сховище хмари буде запроваджене[22].

1.2 Типи хмарних сховищ

Існує декілька типів хмарних сховищ:

1. Приватна хмара. Приватні хмари призначені для розміщення та обробки запитів даних від невеликих компаній або великих компаній. Як правило, цей тип хмарних сервісів є найкращим рішенням для компанії, яка хоче мігрувати всі дані, використовуючи послуги стороннього постачальника послуг у хмарі. Крім того, клієнти хмарних сервісів можуть вибрати два типи приватних хмар. Хмара, що використовує сервер або машину, яку можна знайти в тій же будівлі. Цей сервер розміщує хмару. Іншим варіантом є зовнішнє розміщення обласного сховища. Хост-сервер не знаходиться в тій же будівлі, що і компанія. Наглядові приватні хмари легше контролювати. Машина, яка обробляє всі запити, знаходиться в тій же

будівлі, що й компанія. Таким чином, співробітники можуть легко перевірити, якщо щось піде не так. Однак необхідно мати на увазі, що цей вид послуг краще підходить для великих компаній. Для невеликих підприємств це може бути дорого, оскільки витрати на обслуговування можуть бути досить високими.

2. Публічні хмари. За допомогою публічної хмари вам не потрібно турбуватися про технічне обслуговування або безпеку даних, оскільки всі дані передаються зовні. Публічні хмари легше утримувати та підтримувати, ніж приватні хмари. Крім того, будь-яка особа, яка має відповідні облікові дані, може надати доступ до хмари, або з комп'ютеру, підключеного до Інтернету, або з інших пристроїв, таких як планшет або смартфон.

3. Гібридна хмара. Гібридна хмара поєднує в собі деякі особливості приватних і громадських хмар. Одним з найбільших переваг використання гібридних хмар є більша кількість варіантів налаштування. Є можливість додавати, видаляти або редагувати поточні програми, запущені у вашій хмарі. Якщо необхідно поділитися конфіденційною інформацією, завжди можна вибрати функцію приватної хмари з кращою безпекою. Крім того, є можливість перенести деякі дані на загальнодоступну частину хмари[20].

1.3 Переваги хмарних сховищ

Можна виділити такі основні переваги хмарного сховища:

1. Витрати. Компанії та приватні особи, які використовують хмарні послуги, швидше за все, скоротять операційні витрати, ніж ті, хто все ще використовує внутрішні рішення для хостингу або зовнішні жорсткі диски. Крім того, за даними останніх онлайн-опитувань про хмарне сховище, середня вартість одного гігабайт місця для зберігання становить близько 3 центів.

2. Доступність. Як вже згадувалось, є можливість отримати доступ до всіх файлів, папок, фотографій і відео в хмарі з будь-якої точки світу. Звичайно, якщо у вас є необхідні повноваження та доступ до Інтернету.

3. Відновлення. Одним з найбільших переваг використання хмарних сховищ є те, що у вас завжди буде рішення для резервного копіювання, якщо щось піде не так. Якщо щось трапиться з файлами на комп'ютері, ви завжди можете отримати доступ до хмари та отримати будь-які дані, які могли бути пошкоджені або втрачені.

4. Синхронізація. Якщо внести будь-які зміни до одного або декількох файлів, хмара автоматично синхронізує зміни для всіх дочірніх пристроїв.

5. Підвищена безпека. Більшість постачальників хмарних сховищ, як правило, додають додаткові шари протоколів безпеки. Вони роблять це для того, щоб запобігти потраплянню ваших файлів і папок у неправильні руки або від втрати[21].

Отже можемо зробити висновок, що хмарне сховище це нова концепція в моделях хмарних обчислень, що дозволяє швидше розгортати інформацію, мало керувати та мало контролювати. Є звичайно переваги і недоліки для вибору такого роду послуг. Сьогодні, однак, більшість провайдерів великих хмарних сховищ мають різноманітну пропозицію, коли йдеться про плани зберігання та ціноутворення. Плани та пропозиції варіюються від постачальника до постачальника. Тим не менш, більшість постачальників включають принаймні деякі пакети послуг для домашнього використання, а також плани хмарних сховищ для малих підприємств або великих корпорацій.

Але варто пам’ятати, що користування готовими рішеннями компаній, що надають послуги хмарного середовища можуть бути занадто дорогими. Тому рішенням проблеми зберігання даних можна висунути розробку власної системи хмарного сховища на базі платформ з відкритим API. Тим не менш, тоді необхідно

буде зробити великий акцент на організації контрольованого доступу до даних, що знаходяться у власній системі хмарного сховища.

Висновки до розділу

У даному розділі було проаналізовано проблеми та переваги хмарних сховищ. Також було описано основні види хмарних сховищ, що на сьогодні існують. Акцент було зроблено на тому, що є необхідність створення власного рішення щодо організації хмарного сховища.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ЩОДО ФАЙЛОВИХ СИСТЕМ У ХМАРНОМУ СЕРЕДОВИЩІ

На сьогоднішній день можна помітити, що технології зі зберігання даних невпинно рухаються вперед. Нещодавно люди могли лише думати про те, як вмістити всі дані, які вони мають, на дискету з неймовірно обмеженою пам’яттю. З часом місткість фізичних накопичувачів збільшувалась зі швидкістю геометричної прогресії. Розміри накопичувачів зменшуються та стають більш зручними для використання[1]. Але потроху люди починають рухатись в сторону хмарних технологій, оскільки дана технологія має багато переваг.

2.1 Огляд існуючих рішень хмарних сховищ

У цьому розділі пропонується розглянути існуючі рішення у напрямі хмарних сховищ.

2.1.1 Хмарна платформа Microsoft Azure

«Microsoft Azure» («Майкрософт Ежур») — це хмарна платформа та інфраструктура корпорації Microsoft, призначена для розробників застосунків хмарних обчислень (англ.cloud computing) і покликана спростити процес створення онлайнових додатків[3].

Платформа Windows Azure складається з таких компонентів:

1. Windows Azure — надає середовище виконання для додатків, заснованих на операційних системах, та на Windows Server, а також місця для зберігання даних.
2. Обчислення — відповідає за обчислення розміщення додатків.

3. Зберігання — відповідає за зберігання даних в хмарі.
4. SQL Azure — надає можливість використовувати реляційну базу даних для запуску в хмарі.
5. Список сервісів Azure налічує багато служб, деякі з них перераховані нижче:
6. Служби зберігання даних надають REST та SDK API для зберігання та доступу до даних у хмарі.
7. Table Service надає можливість програмам зберігати структурований текст у розподілених колекціях, доступ надається за ключем розділу та первинним ключем.
8. Blob Service дозволяє зберігати неструктурований текст та бінарні дані як blob'и, які доступні за HTTP(S) адресою. , також надає способи контролю доступу до даних.
9. Queue Service забезпечує асинхронну комунікацію програм.

Azure - це швидка, гнучка і доступна платформа, і її ціноутворення та можливості роблять його одним з найкращих публічних хмарних сервісів. Дана технологія може запропонувати покращену реалізацію резервного копіювання та відновлення. Як облачне рішення, Azure є дуже гнучкою технологією - вона може створювати резервні копії даних майже на будь-якій мові, на будь-якій ОС і з будь-якого місця. Крім того, ви визначаєте частоту та обсяг розкладу резервного копіювання (щодня, щотижня, щомісяця тощо)[15].

Стрічкове резервне копіювання має час і місце, але воно має обмежені можливості як автономне рішення для резервного копіювання та аварійного відновлення. Відновлення сайту Azure може підвищити резервну копію стрічки за допомогою реплікації за межами об'єкта, мінімального технічного обслуговування на місці, до дев'яноста дев'яти років зберігання даних, а також мінімальних операційних витрат. Резервне копіювання Azure зберігає три копії даних у трьох різних місцях у центрі обробки даних, а потім ще три копії у віддаленому центрі даних Azure, тому вам ніколи не доведеться турбуватися про втрату даних.

Якщо знаходитись у віртуальному середовищі Windows, вбудована інтеграція Azure для додаткового резервного копіювання буде швидким і безболісним рішенням. Відновлення сайтів Azure інтегрується з архітектурами System Center і HyperV, створюючи надійну і безперервну згуртованість між Azure, System Center і HyperV.

Також незалежно від того, чи є необхідність в платформі для хостингу, розробки чи керування веб- або мобільним додатком, Azure робить ці програми автономними та адаптивними з керуванням виправленнями та AutoScale.

Завдяки автоматичному керуванню виправленнями для віртуальних машин є можливість витрачати менше часу на керування інфраструктурою та зосереджувати увагу на покращенні програм. Azure також постачається з безперервною підтримкою розгортання, яка дозволяє упорядкувати поточні оновлення коду.

Функція AutoScale - це функція, вбудована в Azure Web Apps, яка автоматично налаштовує ресурси на основі веб-трафіку клієнта, тому завжди є забезпеченяя потрібними ресурсами, коли кількість трафіку висока.

За допомогою Azure можна безперешкодно пов'язувати веб-додаток із місцевим додатком. Підключення додатків в обох місцях дозволяє співробітникам і партнерам надійно отримати доступ до ресурсів у брандмауері - ресурсах, які інакше було б важко отримати ззовні.

Azure може інтегруватися з Active Directory для доповнення можливостей ідентифікації та доступу - це надає доменній системі імен глобальний доступ, централізоване управління та надійну безпеку.

За допомогою Azure можна глобально поширювати середовище Active Directory, у якому включено пряме підключення. Жоден інший постачальник послуг хмари не має можливості розширювати охоплення контролера домену та консолідувати управління AD, як Azure.

Якщо в наявності декілька місць розташування або використовується наявні програми або хмарні програми, такі як Office 365, інтеграція Active Directory з Azure

стане центральним інструментом для керування та збереження доступу до всіх цих інструментів.

Azure також дозволяє використовувати багатофакторну аутентифікацію, додаючи новий рівень безпеки до даних і додатків з нульовою турботою для користувачів. Також є можливість легко реалізувати єдину реєстрацію для хмарних додатків Windows, Mac, Android і iOS.

Масштабованість, гнучкість і безпека Microsoft Azure робить його відповідним ресурсом для компаній, що рухаються до рішень IoT. Також є можливість підключити свої пристрої до хмари, використовуючи рішення, які інтегруються з існуючою інфраструктурою.

2.1.2 Хмарна платформа Google Cloud Platform

Google Cloud Platform (GCP) - це сукупність обчислювальних ресурсів Google, які доступні через послуги для широкої громадськості як пропозицію публічного хмарного ресурсу.

Ресурси GCP складаються з фізичної апаратної інфраструктури - комп'ютерів, жорстких дисків, твердотільних накопичувачів і мереж - містяться в глобально розповсюджених дата-центрах Google, де будь-який з компонентів розроблений з використанням шаблонів, аналогічних тим, які доступні в проекті Open Compute.

Це обладнання надається клієнтам у вигляді віртуалізованих ресурсів, таких як віртуальні машини (VM), як альтернатива клієнтам, які будують і підтримують власну фізичну інфраструктуру[7].

Публічна пропозиція хмарних програм, програмні та апаратні продукти надаються як інтегровані послуги, що забезпечують доступ до базових ресурсів. GCP пропонує більше 50 послуг, включаючи таку інфраструктуру у ролі сервісу (IaaS), платформу у ролі сервісу (PaaS), а також пропозиції програмного

забезпечення у ролі сервісу (SaaS) у категоріях Compute, Storage & Databases, Networking, Big Data, Machine Learning, Identity із засобами для розробника.

Ці послуги можуть використовуватися самостійно або в поєднанні для розробників та IT-фахівців для створення власної обласної інфраструктури.

Програма GCP розміщена на тій же базовій інфраструктурі, що і сам Google.

Кожна з служб і ресурсів Google Cloud Platform може бути зональною, регіональною або керованою Google у кількох регіонах.

Зона - це область розгортання ресурсів у межах регіону. Зони відокремлені один від одного, щоб запобігти поширенню відключень між ними, тому кожна зона вважається окремою областю в межах регіону.

Зональні ресурси функціонують в межах однієї зони; якщо зона стає недоступною, всі її ресурси стають недоступні до відновлення служби. Регіональні ресурси розгортаються з надмірним розподілом по зонах регіону. Google має багаторегіональні послуги (Google App Engine, Google Datastore, Google Cloud Storage, Google BigQuery), які є надлишковими та розповсюджуються в межах та між регіонами.

Усі екземпляри служби GCP налаштовані таким чином, що події обслуговування є прозорими для програм і робочих навантажень через міграцію в реальному часі. Жива міграція переміщує запущені екземпляри віртуальних машин у спосіб, який виконується.

Хмарна платформа Google забезпечує надійну та масштабовану інфраструктуру для розробників для створення, тестування та розгортання додатків. Вона охоплює сервіси прикладних програм, зберігання даних та обчислень для бекенда, мобільних та веб-рішень. Більше чотирьох мільйонів програм довіряють і використовують платформу.

Google намагається максимально простим сервером і використовує просту файлову систему. Ця система є основою хмарної платформи Google. Він обробляє

запити інформації через основні команди, такі як запис, читання і відкриття. Це розподілена система обчислень.

Файлова система складається з мереж, які також відомі як кластери. Кожному кластеру виділено один головний сервер. Система координує запити даних через ці основні сервери. Дії користувачів перетворюються на запити даних, коли вони взаємодіють з хмарою та інформацією, що зберігається на ній. Написання або форматування нових даних або інших простіших дій, таких як перегляд файлу, належать до цих запитів даних[16].

Комп'ютер користувача надсилає запити даних до інших комп'ютерів і діє як клієнт. Комп'ютер Google, який має дані, отримує повідомлення від головного сервера, який приймає запит. Комп'ютери, які мають дані, називаються chunk-серверами. Запитувана інформація не проходить через головний сервер і надсилається безпосередньо клієнту з сервера порції.

Трохи складніше вносити зміни до даних у хмарі, оскільки Google зберігає декілька копій даних. Насамперед, головний сервер отримує запит на запис. Сервер вибирає chunk-сервер, який має відповідні дані. Це відоме як первинна репліка chunk-сервера. Клієнт дізнається про місця розташування всіх реплік chunk-серверу через головний сервер.

Коли користувач вносить будь-які зміни, вони передаються першій репліці chunk-серверу, до якого може з'єднатися комп'ютер користувача. Потім запит на зміну переміщується через систему до всіх реплік chunk-серверу, включаючи первинний. Первинний chunk-серверами робить зміни і інструктує всім реплікам робити те ж саме.

Переваги Google Cloud:

1. Вища продуктивність здобувається завдяки швидкому доступу до інновацій: системи Google можуть ефективно поширювати оновлення та надавати функціональні можливості щотижня або навіть швидше.

2. Менше порушень викликається тим, що користувачі приймають нову функціональність: замість великих руйнівних пакетів змін Google забезпечує керовані поліпшення в безперервному потоці.

3. Працівники можуть працювати з будь-якого місця: вони можуть отримати повний доступ до інформації через пристрої з будь-якої точки світу через веб-додатки, засновані на хмарі Google.

4. Google Cloud дозволяє швидко співпрацювати. Багато користувачів можуть робити свій внесок та отримувати доступ до проектів одночасно з тим, як дані зберігаються на хмарі, а не на комп'ютерах.

5. Менше даних потрібно зберігати на вразливих пристроях: мінімальні дані зберігаються на комп'ютерах, які можуть бути скомпрометовані, після того, як користувач перестане користуватися веб-додатками на хмарі.

6. Клієнти отримують більш високу швидкість та надійність: якщо центр даних недоступний з певних причин, система негайно повертається на вторинний центр без будь-якого переривання сервісу, який буде видно користувачам.

7. Контроль і гнучкість, доступні для користувачів: вони мають контроль над технологіями і даними і мають право власності на свої дані в програмах Google. Якщо вони вирішать більше не використовувати цю послугу, вони зможуть отримати свої дані з хмари Google.

8. Економія масштабу Google дозволяє споживачам витрачати менше: Google мінімізує накладні витрати та консолідує невелику кількість конфігурацій сервера.

9. Надзвичайна безпека: Google наймає провідних експертів з безпеки у всьому світі.

Хмара Google має величезний вплив, особливо в світі розробки веб-і мобільних додатків, де вона спростила і спростила розробників. Проте, її залежність від підключення до Інтернету може в майбутньому дещо завадити його прогресу.

2.1.3 **Хмарна платформа AWS**

Amazon Web Services (AWS) - це безпечна платформа хмарних сервісів, яка пропонує обчислювальну потужність, сховище баз даних, доставку контенту та іншу функціональність[17].

AWS запустила в 2006 році внутрішню інфраструктуру, яку Amazon.com побудував для обробки своїх онлайн-роздрібних операцій. AWS була однією з перших компаній, що впровадила модель хмарних обчислень, що надає користувачам можливість хмарних обчислень та зберігань.

Amazon Web Services надає послуги з десятків центрів обробки даних, які поширюються по зонах доступності (ЗД) у регіонах по всьому світу. ЗД представляє розташування, яке зазвичай містить декілька фізичних центрів даних, тоді як регіон є збіркою ЗД в географічній близькості, з'єднаної мережевими посиланнями з низькою затримкою. Клієнт AWS може розкручувати віртуальні машини (ВМ) і повторювати дані в різних ЗД, щоб досягти високонадійної інфраструктури, стійкої до збоїв окремих серверів або всього центру обробки даних.

Amazon Elastic Compute Cloud (EC2) надає віртуальні сервери - звані екземпляри - для обчислювальної потужності. Послуга EC2 пропонує десятки типів екземплярів з різними потужностями та розмірами, пристосовані до конкретних типів робочих навантажень та додатків, таких як робота з об'ємними та прискореними обчислювальними пристроями. AWS також надає інструмент для автоматичного масштабування для динамічного збільшення потужності для збереження стану та продуктивності екземпляру .

Контейнерна служба Amazon EC2 і реєстр контейнерів EC2 дозволяють клієнтам працювати з контейнерами Docker і зображеннями на платформі AWS. Розробник може також використовувати AWS Lambda для безсерверних функцій, які автоматично виконують код для програм і служб, а також AWS Elastic Beanstalk

для PaaS. AWS також включає в себе Amazon Lightsail, який надає віртуальні приватні сервери, і AWS Batch, який обробляє ряд завдань.

Amazon Simple Storage Service (S3) забезпечує масштабоване сховище об'єктів для резервного копіювання даних, архівування та аналітики. Фахівці з інформаційних технологій зберігають дані та файли як об'єкти S3, які можуть містити до 5 Гб у S3, щоб підтримувати їх організацію.

Amazon Elastic Block Store забезпечує томи зберігання на рівні блоків для постійного зберігання даних для використання з екземплярами EC2, в той час як Amazon Elastic File System пропонує керовані хмарні файлові сховища.

Також можна мігрувати дані до хмари за допомогою пристроїв транспортування сховищ, таких як AWS Snowball та Snowmobile, або використовувати шлюз AWS Storage для того, щоб дозволити локальним програмам доступу до даних у хмарі.

AWS надає послуги керованих баз даних через свою службу реляційних баз даних Amazon, яка включає опції для Oracle, SQL Server, PostgreSQL, MySQL, MariaDB і власну базу даних високої продуктивності під назвою Amazon Aurora. AWS пропонує керовані бази даних NoSQL через Amazon DynamoDB.

Клієнт AWS може використовувати Amazon ElastiCache і DynamoDB Accelerator як кеші для даних в оперативній пам'яті для програм у реальному часі. Amazon Redshift пропонує сховище даних, що полегшує аналітикам даних виконання завдань бізнес-аналітики.

AWS включає в себе різні інструменти та послуги, призначені для допомоги користувачам мігрувати їх додатки, бази даних, сервери та дані на свою публічну хмару. Концентратор міграції AWS надає місцезнаходження для відстеження та управління міграціями у хмару. Після переходу до хмари, EC2 Systems Manager допомагає IT команді налаштувати локальні сервери та екземпляри AWS.

Amazon також співпрацює з кількома виробниками технологій, які спрощують розгортання гібридної хмари. VMware Cloud на AWS привносить технологію

технологічних центрів з VMware в хмару AWS. Red Hat Enterprise Linux для Amazon EC2 є продуктом іншого партнерства, поширюючи операційну систему Red Hat на хмару AWS.

Віртуальна приватна хмара Amazon (VPC) надає адміністратору контроль над віртуальною мережею для використання ізольованого розділу хмари AWS. AWS автоматично забезпечує нові ресурси в VPC для додаткового захисту.

Адміністратори можуть збалансувати мережевий трафік за допомогою засобів балансування навантаження AWS, включаючи балансування навантаження прикладних програм і балансування мережевого навантаження. AWS також надає систему доменних імен Amazon Route 53, яка спрямовує кінцевих користувачів до додатків.

IT-фахівець може встановити спеціальне підключення від локального центру обробки даних до хмари AWS за допомогою AWS Direct Connect.

Розробник може скористатися перевагами інструментів командного рядка AWS і комплектів програмного забезпечення для розгортання програм і служб і керування ними. Інтерфейс командного рядка AWS є власним кодом інтерфейсу Amazon. Розробник може також використовувати інструменти AWS для Powershell для керування хмарними службами з середовищ Windows і AWS Serverless Application Model для моделювання середовища AWS для перевірки функцій Lambda. AWS SDK доступні для різних платформ і мов програмування, включаючи Java, PHP, Python, Node.js, Ruby, C ++, Android і iOS.

Шлюз API Amazon дозволяє розробникам створювати, керувати та контролювати спеціальні API, що дозволяє користувачам отримувати доступ до даних або функціональних можливостей з серверних служб. Шлюз API керує тисячами одночасних викликів API.

AWS також надає послугу пакувального медіатрансляції, Amazon Elastic Transcoder і службу, яка візуалізує робочі процеси для додатків на основі мікросервісів - AWS Step Fun.

Команда розробників може також створювати безперервну інтеграцію і безперервні конвеєри доставки з такими службами, як AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy і AWS CodeStar. Розробник може також зберігати код в сховищах Git за допомогою AWS CodeCommit і оцінювати продуктивність додатків на основі мікросервісів з AWS X-Ray.

Адміністратор може керувати та відстежувати конфігурацію ресурсів хмари за допомогою AWS Config та AWS Config Rules. Ці інструменти, разом з AWS Trusted Advisor, можуть допомогти IT-команді уникнути неправильного налаштування та непотрібної розгортання ресурсів хмари.

AWS надає декілька засобів автоматизації у своєму портфелі. Адміністратор може автоматизувати надання інфраструктури за допомогою шаблонів AWS CloudFormation, а також використовувати AWS OpsWorks та Chef для автоматизації конфігурацій інфраструктури та системи.

Клієнт AWS може відстежувати стан ресурсів і додатків за допомогою Amazon CloudWatch і AWS Personal Health Dashboard, а також використовувати AWS CloudTrail для збереження активності користувачів і викликів інтерфейсу прикладного програмування (API) для аудиту.

AWS надає широкий спектр послуг для безпеки в хмарах, включаючи AWS Identity and Access Management (IAM), що дозволяє адміністраторам визначати і керувати доступом користувачів до ресурсів. Адміністратор може також створити директорію користувача з каталогом Amazon Cloud або з'єднати ресурси хмари з існуючою службою Microsoft Active Directory з службою каталогів AWS. Окрім того, AWS Organizations надає можливість бізнесу створювати та керувати політиками для декількох облікових записів AWS.

Провайдер хмари також ввів інструменти, які автоматично оцінюють потенційні ризики безпеки. Amazon Inspector аналізує середовище AWS для вразливостей, які можуть вплинути на безпеку та відповідність. Amazon Macie використовує технологію машинного навчання для захисту чутливих даних у хмарі.

AWS також включає засоби та послуги, що забезпечують програмне та апаратне шифрування, захищають від DDoS-атак, забезпечують сертифікати Secure Sockets Layer і Transport Layer Security і фільтрують потенційно небезпечний трафік для веб-додатків.

AWS включає в себе безліч великих аналітичних даних і сервісів прикладних програм. Amazon Elastic MapReduce пропонує структуру Hadoop для обробки великих обсягів даних, в той час як Amazon Kinesis надає кілька інструментів для обробки та аналізу потокових даних.

AWS Glue - це служба, яка обробляє екстракцію, перетворення та завантаження, а служба Amazon Elasticsearch дозволяє команді виконувати моніторинг додатків, аналіз журналів та інші завдання за допомогою інструменту з відкритим вихідним кодом Elasticsearch.

Для запиту даних аналітик може використовувати Amazon Athena для S3, а потім візуалізувати дані за допомогою Amazon QuickSight.

AWS пропонує широкий спектр платформ для розробки і доставки AI, а також пакетні програми на основі AI. Набір інструментів Amazon AI включає Amazon Lex для технології голосових і текстових чатів, Amazon Polly для перекладу текстів у мову та Amazon Rekognition для аналізу зображень та обличчя. AWS також забезпечує технологію для розробників для створення смарт-додатків, які спираються на технології машинного навчання та складні алгоритми.

За допомогою AIS Deep Learning AMI розробники можуть створювати та навчати користувальницькі моделі AI з кластерами GPU або оптимізованими для обчислень екземплярами. AWS також включає в себе глибокі рамки для розвитку навчання для MXNet і TensorFlow.

З боку споживачів, технології AWS надають послуги Alexa Voice Services, а розробник може використовувати набір навичок Alexa для створення голосових програм для пристроїв Echo.

Веб-служби Amazon мають ряд опцій SaaS для підвищення продуктивності бізнесу. Послуга Amazon Chime дозволяє здійснювати онлайн-зустрічі, дзвінки та текстові чати між пристроями. Бізнес також може скористатися перевагами Amazon WorkDocs, служби зберігання та обміну файлами та службою електронної пошти Amazon WorkMail з функціями календаря.

Служби настільних і потокових додатків включають в себе Amazon WorkSpaces, віддалену платформу для настільного настільного обслуговування і Amazon AppStream, службу, яка дозволяє розробнику передати настільну програму від AWS до веб-браузера кінцевого користувача.

AWS також має цілий ряд послуг, які дозволяють розгортати Інтернет-речі (IoT). Служба AWS IoT надає платформу для керування пристроями IoT та передачею даних іншим сховищам AWS та службам бази даних. Кнопка AWS IoT забезпечує обладнання для обмеженої функціональності IoT, і AWS Greengrass надає обчислювальні можливості AWS пристроям IoT.

Отже, хмара Amazon - це безпечний сервіс, який може допомогти вам знизити витрати на інфраструктуру, наприклад на сервери, а також заощадити на кількості співробітників, необхідних для побудови та підтримки інфраструктури.

2.1.4 **Хмарна платформа Heroku**

Heroku - платформа хмарних сервісів, популярність якої за останні роки зросла. Heroku настільки проста у використанні, що вона є найкращим вибором для багатьох проектів розвитку.

Особлива увага приділяється підтримці програм, орієнтованих на клієнтів, вона дозволяє легко розробляти та розгортати програми. Оскільки платформа Heroku управляє апаратними засобами та серверами, підприємства, які використовують Heroku, можуть зосередитися на вдосконаленні своїх додатків, а не на інфраструктурі, яка їх підтримує.

Heroku, і приклад рішення платформи як сервіса, як правило, вважається простим у використанні. Але це є найбільш вигідним для підприємств у конкретних ситуаціях. У Heroku є безкоштовна модель обслуговування для невеликих проектів, але існують багаторівневі пакети послуг для випадків, коли необхідно вирішувати більш складні бізнес-потреби.

Сервісно-орієнтовані архітектури - суть хмарних обчислень - можуть бути розділені їх метою.

Найбільш примітними категоріями є інфраструктура як сервіс (IaaS), платформа як сервіс (PaaS) і програмне забезпечення як сервіс(SaaS).

Архітектура IaaS забезпечує зберігання, мережі, обчислювальні потужності та інші базові обчислювальні ресурси, де користувачі послуг можуть розгортати та запускати довільне програмне забезпечення. Програмне забезпечення може включати операційні системи (OS) та програми.

У цій моделі користувачі взагалі не мають контролю над базовою хмарою, а керують операційними системами, сховищами, розгорнутими додатками та, можливо, деякими частинами мережевих компонентів.

Приклади архітектур IaaS включають веб-служби Amazon і Microsoft Azure, що було розглянуто вище.

Архітектура PaaS зазвичай включає операційні системи, середовища виконання мов програмування, бібліотеки, бази даних, веб-сервери і підключення до деяких платформ.

Користувачі послуг PaaS можуть розгортати створені споживачем або придбані програми на хмарну інфраструктуру. Підприємства, які використовують PaaS, мають контроль над розгорнутими додатками, але мають обмежений доступ до налаштувань конфігурації для середовища, де розміщено прикладні програми. Вони також мають обмежений доступ до конфігураційних параметрів ресурсів, що надаються PaaS.

Користувачі не керують базовою хмарою, включаючи мережу, сервери, операційні системи або сховище. Натомість їх розробники зосереджуються на управлінні програмами, що підтримуються розгорнутою платформою.

Google App Engine, Apache Stratos і AWS Elastic Beanstalk є прикладами пропозицій PaaS, і кожна служба спрощує роботу розробників, розробляючи, перевіряючи та розгортаючи свої програми.

Послуги SaaS надають користувачам можливість користуватися програмами провайдера, які працюють на хмарі. Програми доступні з різних клієнтських пристроїв через клієнтський інтерфейс, наприклад веб-браузер, мобільні програми або інтерфейс програми.

У деяких випадках користувачі мають обмежений доступ до параметрів конфігурації додатків, але вони не керують базовою хмарою, включаючи мережу, сервери, операційні системи, сховище або навіть можливості програми.

Доступ до послуг у групі SaaS набагато більш обмежений, ніж доступ до груп послуг PaaS або IaaS.

Переваги, які надаються цими послугами, включають можливість простого масштабування, доступу, управління, інтеграції, розподілу, обслуговування та тарифікації. Керівники підприємств можуть легко оцінити свої витрати, оскільки багато послуг оплачуються щомісяця.

Це сприяє простому прогнозуванню майбутніх витрат, коли технічні групи аналізують, планують і розробляють проекти для клієнтів.

Heroku підтримує багато мов програмування. Heroku, одна з перших хмарних платформ, розробляється з червня 2007 року, коли вона підтримувала тільки мову програмування Ruby.

Але тепер Heroku також підтримує Java, Node.js, Scala, Clojure, Python, PHP і Go. Це означає, що різноманітні розробники можуть звернутися до Heroku для недорогого способу масштабування своєї програми, незалежно від їхньої кращої мови розвитку.

Платформа хмарних сервісів Heroku базується на керованому контейнері (який називається dynos в рамках парадигми Heroku) з інтегрованими послугами даних і потужною екосистемою для розгортання і запуску сучасних додатків.

Dyno є ізольованими, віртуалізованими контейнерами Linux, призначеними для виконання коду на основі заданої користувачем команди.

Heroku також надає спеціальні комплекти зборки, де розробники можуть розгортати програми на будь-якій іншій мові програмування. З цієї причини Heroku називають платформою для поліглотів. Це дозволяє розробнику створювати, запускати і масштабувати програми подібним чином на всіх мовах програмування.

Поліморфізм і масштабованість є причинами, чому Heroku часто розглядається як бажана платформа серед розробників.

Програми, які виконуються на Heroku, зазвичай мають унікальні доменні імена, які використовуються для маршрутизації HTTP-запитів до правильного контейнера. Програми як сервіси використовують контейнери додатків. Контейнери призначені для упаковки та виконання послуг. Кожен з контейнерів додатків є розумним контейнером на надійній, повністю керованій середовищі виконання. Контейнери додатків - називаються "dynos" у контексті платформи Heroku - поширюються по «сітці dyno». Менеджер Dyno підтримує і управляє створеним dyno.

Все це означає, що, оскільки Heroku керує і запускає програми, немає необхідності керувати операційними системами або іншими внутрішніми конфігураціями системи.

Heroku дозволяє розробникам масштабувати програми миттєво.

Це досягається або збільшенням числа dynos, або зміною типу dyno, в якому працює додаток. Коли додаток може легко масштабуватися, користувач завжди може розраховувати на більшу швидкість при використанні цієї програми.

Простий спосіб масштабування додатків робить роботу з Heroku легкою та зручною.

Проекти, створені в Heroku, прив'язані до сховищ у GitHub. Інтеграція Heroku з GitHub забезпечує автоматизоване створення і розгортання останньої версії коду.

GitHub є надійним сховищем вихідного коду, тому інтеграція Heroku з GitHub та іншими інструментами допомагає розробникам оптимізувати свої зусилля та заощадити час і гроші учасників у ході проектів розвитку.

Порівняно з іншими хмарними платформами PaaS, Heroku має одну з найкраще налаштованих служб хмарних обчислень, незалежно від того, чи відбувається міграція до хмари, чи зберігаються файли або розгортають програмне забезпечення через хмару, розуміння того, як правильно використовувати служби хмарних обчислень продовжує домінувати в розмові в IT-індустрії.

Heroku - це хмарна платформа, яка дозволяє компаніям створювати, доставляти та контролювати прості додатки швидко і без великої кількості турбот в інфраструктурі. Платформа підходить для стартап компаній, проектів і клієнтів, які хочуть отримати тест першої версії свого продукту, перш ніж зайнятись інвестиціями в обладнання та інфраструктуру.

Для компаній на ранніх етапах розвитку бізнесу та продуктів Heroku може бути правильним рішенням для перевірки ідей та якості раннього кодування.

2.2 Огляд існуючих рішень аутентифікації користувача

Одним з основних способів захисту ресурсу є гарантія того, що користувач, який користується сервісом дійсно має на це право. Цей процес перевірки облікових даних і забезпечення їхньої справжності називається аутентифікацією. Дуже важливо вибрати коректний спосіб аутентифікації для хмарного сховища. Спосіб аутентифікації повинен бути максимально безпечним та зручним. Пропонується розглянути два способи: Spring Security Authentication та The OAuth 2.0 Authorization Framework.

2.2.1 **Аутентифікація Spring Security**

Spring Security має ряд фільтрів сервлетів (ланцюг фільтрів). Коли запит доходить до сервера, його перехоплює ця серія фільтрів[18]. У реактивному світі, за допомогою нового веб-середовища Spring WebFlux, фільтри записуються зовсім інакше, ніж традиційні фільтри, такі як ті, що використовуються в рамках веб-додатків Spring MVC. Основний механізм залишається незмінним для обох.

Виконання коду фільтра Servlet у ланцюжку фільтрів продовжує проходити далі, поки не буде досягнутий правильний фільтр. Після досягнення правильного фільтра аутентифікації на основі використовуваного механізму аутентифікації він дістає надані облікові дані , найчастіше ім'я користувача та пароль , від користувача.

Використовуючи наведені значення (при використанні імені та паролю), фільтр UsernamePasswordAuthenticationFilter створює об'єкт аутентифікації. На діаграмі UsernamePasswordAuthenticationToken створюється з іменем користувача та паролем, наданим у кроці 2. Об'єкт аутентифікації, створений на кроці 2, використовується для виклику методу аутентифікації в інтерфейсі AuthenticationManager.

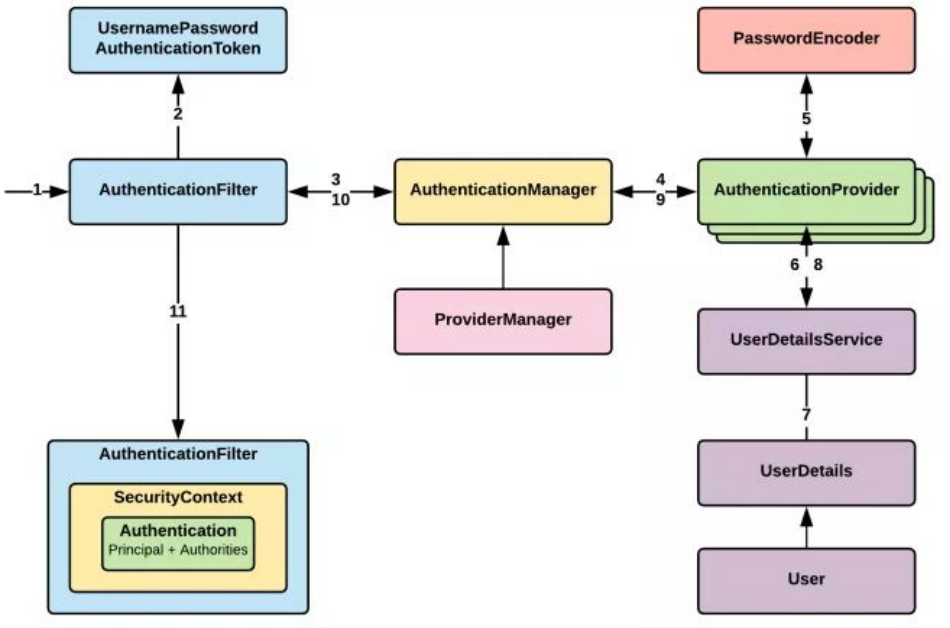


Рисунок 2.1 – Діаграма роботи аутентифікації Spring Security

Запит проходить через різних провайдерів і одночано намагається аутентифікувати запит. Існує ряд інтерфейсів AuthenticationProvider як частина Spring Security.

На діаграмі вище, AuthenticationProvider вимагає відомостей про користувача (деякі провайдери вимагають цього, але деякі не мають), які надаються в UserDetailsService.

UserDetailsService витягує UserDetails і реалізує інтерфейс користувача, використовуючи надане ім'я користувача.

Якщо всі дані підтверджуються, Spring Security створює повністю заповнений об'єкт аутентифікації (аутентифікація: флаг правда, наданий список прав і ім'я користувача), які містять різні необхідні деталі. Об'єкт аутентифікації зберігається у об'єкті SecurityContext за допомогою фільтра для подальшого використання[9].

Дані процедури являються базовими та найменшими необхідними умовами для проходження аутентифікації користувачем.

2.2.2 **Аутентифікація The OAuth 2.0**

OAuth означає відкриту авторизацію. Це вільний і відкритий протокол, побудований на стандартах IETF і ліцензіях від Open Web Foundation. Це дозволяє користувачам ділитися своїми приватними ресурсами з третьою стороною, зберігаючи при цьому свої власні облікові дані. Ці ресурси можуть бути фотографіями, відео, списками контактів, можливостями розташування та виставлення рахунків тощо, і зазвичай зберігаються у іншого постачальника послуг. OAuth виконує це, надаючи токен запиту додаткам, коли користувач затверджує доступ. Кожен токен надає обмежений доступ до певного ресурсу за певний період. OAuth2 підтримує «делеговану аутентифікацію», тобто надає доступ іншому користувачеві або програмі для виконання дій від вашого імені.

OAuth вимагає, щоб клієнт реєструвався на сервері авторизації. Сервер авторизації запитує деякі основні відомості про клієнта, такі як ім'я, redirect_uri (URL, на який сервер перенаправлення перенаправляє, коли власник ресурсу надає дозвіл) і повертає клієнтські облікові дані (ідентифікатор клієнта, клієнт-секрет) клієнту. Ці облікові дані є критичними для захисту автентичності запитів при виконанні таких операцій, як обмін кодами авторизації для токенів доступу та оновлення токенів доступу.

Щоб отримати токен доступу, клієнт отримує авторизацію від власника ресурсу. Авторизація виражається у вигляді дозволу на авторизацію, який клієнт використовує для запиту токену доступу. OAuth2 має власні типи доступу та надає механізм розширення для визначення додаткових типів доступу. OAuth2 визначає чотири основних типи доступу: доступ по коду авторизації, неявний доступ,доступ по обліковим даним власника ресурсу та доступ обліковим даним клієнта:

1. Доступ по коду авторизації: цей тип доступу оптимізований для конфіденційних клієнтів (сервера веб-додатків). Потік коду авторизації не відкриває токен доступу браузеру власника ресурсу. Замість цього авторизація здійснюється за допомогою посередницького «коду авторизації», який передається через браузер. Цей код необхідно обміняти на токен доступу до того, як відбувся виклик захищених API.

2. Неявний доступ: цей тип дозволу підходить для публічних клієнтів. Неявний потік дозволу не пристосовується до токенів оновлення. Якщо сервер авторизації закінчується регулярними токенами доступу, додатку потрібно буде запустити потік авторизації, коли він потребуватиме доступу. У цьому потоці токен доступу негайно повертається клієнту після того, як користувач надає запитану авторизацію. Проміжний код авторизації не є обов'язковим, оскільки він є в дозволі на код авторизації.

3. Доступ по обліковим даним власника ресурсу: підходить у випадках, коли власник ресурсу має довірчі відносини з клієнтом, і власник ресурсу

погоджується надавати клієнту свої облікові дані (ім'я користувача, пароль). Потім клієнт може використовувати ресурси з облікових даних власника, щоб отримати токен доступу з сервера авторизації.

4. Доступ обліковими даними клієнта: цей тип гранту підходить, коли сам клієнт володіє даними і не потребує делегованого доступу від власника ресурсу, або делегований доступ вже надано застосунку поза типовим потоком OAuth. У цьому потоці згода користувача не бере участі. Клієнт обмінюється обліковими даними клієнта, щоб отримати токен доступу[8].

Здійснення викликів API за допомогою токену доступу OAuth 2.0 може спричинити помилки, якщо токен доступу більше не є дійсним, оскільки актуальність токену закінчилась або він був відкликаний. У цьому випадку сервер ресурсів поверне код помилки 4xx. Клієнт може отримати новий токен доступу, використовуючи токен оновлення (який був отриманий, коли код авторизації був обміняний на токен доступу).

Отже даний метод авторизації дає дуже великий спектр можливостей для розробника та власника веб-сервісу щодо отримання інформації про користувача та забезпечення підвищеного контролю доступу до системи.

Висновки до розділу

В цьому розділі було розглянуто існуючі рішення щодо реалізації хмарного сховища та розгортання системи роботи з даним хмарним сховищем.

Отже для виконання дипломної роботи було обрано саме платформу Heroku, адже ця платформа має всі переваги для реалізації кафедрального хмарного сховища з контрольованим доступом до самого сховища, та до файлів, що у ньому знаходяться. Для аутентифікації було обрано The OAuth 2.0 Authorization Framework, оскільки даний фреймворк відрізняється підвищеною зручністю використання та безпекою при розробці власного веб-додатку.

3 ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Проаналізувавши поставлені задачі, було вирішено реалізувати систему хмарного сховища у вигляді веб-додатку. Основною перевагою веб-додатку є кросплатформеність: немає необхідності прив’язуватись до пристрою, на якому буде використовуватися дана система[2]. Також дане рішення не потребує додаткових дій від користувача, бо йому не потрібно встановлювати та налаштовувати додаток на своєму комп’ютері. Достатньо перейти за посиланням у браузері, пройти авторизацію та почати користуватися системою.

Було використано середовище розробки IntelliJ IDEA.

Для розробки авторизованого доступу до системи та для контролю доступу до файлів всередині системи було використано мову Java, фреймоворки Spring Boot та Spring Security. Основним інструментом авторизації є технологія OAuth2.

У якості бази даних було використано PostgreSQL.

3.1 **Необхідність писати чистий код**

Не завжди єдиною задачею розробника є реалізування певного функціоналу згідно із поставленою задачею. Не потрібно забувати, що запорукою успіху у процесі розробки є написання якісного та зрозумілого коду.

Коли справа доходить до написання програмного забезпечення, часто при запуску та написанні перших рядків класу може бути трохи більше коду, ніж потрібно. Наприклад, відкритий метод у службі може містити деякі змінні, які не є необхідними, блок логіки або "if" оператори, які слід виокремити в окремий метод. Наприклад, результат методу може бути використаний безпосередньо в коді, створення змінної і присвоєння результату цій змінної не є обов'язковим.

Необхідно зменшувати кількість коду. Видалити невикористані змінні. Вилучити коментарі також - код повинен бути очевидним.

Також потрібно робити акцент на тому якої довжини метод, що відповідає за деякий функціонал. Якщо він має більше 20 рядків, багато операцій "if", циклів та інше, тоді необхідно виокремити його до іншого методу з належним ім'ям, виходячі з якого буде одразу зрозуміло, що саме робить метод, не аналізуючи логіку. Це допомагає провести легку, прочитану структуру методу. Завжди краще прочитати один метод, що містить 20 рядків, що викликають 10 різних методів, ніж метод, що містить 400 рядків без логічної назви, що супроводжується великим коментарем.

Варто витратити деякий час, щоб знайти своє ім'я для класу, методу, змінної, яку ми створюємо. Це врятує наших колег багато часу. Не використовуйте абревіатури, імена повинні бути простими і самороз'яснювальними. Наприклад, "usersAction" - це незрозуміла назва методу - щоб знати, що це означає, ви повинні дивитися всередину тіла методу. Навпаки, метод з самороз'яснювальним ім'ям може бути "saveUserInDataBaseAndSendAnEmail". Це ім'я точно говорить про те, що саме робить метод. Усередині можна очікувати виклики окремих методів з іменами: "saveUser", "sendAnEmailAboutSavedUser".

Назва змінної повинна розповісти, які дані вона тримає або яка служба ховається за нею.

Лише за назвою методу необхідно точно знати, що він робить, не занурюючись у логіку.

Клас, повинен бути названий відповідно до домену (частина системної логіки, за яку він відповідає, наприклад, домен може бути платежами, рахунками тощо). Якщо клас почне рости більш ніж на 200 + рядків, необхідно його розділити.

Якщо писати чистий код, то це допоможе майбутній розробці і співробітникам, що також працюють над проектом. Зменшується вартість обслуговування програми, що розробляється. Спрощується оцінка часу, необхідного

для нових функцій. Полегшується виправлення помилок. По суті, полегшується робота над проектом у всіх напрямках.

“Якщо ви хочете, щоб ваш код легко було написати, зробіть його легким для читання.” - Роберт Мартін.

3.2 Середовище розробки IntelliJ IDEA

IntelliJ IDEA - це інтегроване середовище розробки Java (IDE). Використовується для розробки програмного забезпечення. Він розроблений компанією JetBrains. Вона поставляється під ліцензією apache2 "Community edition", а також "власним комерційним виданням". Це найкращі доступні середовища розробки Java. Вона надає можливості, такі як вдосконалена кодової навігації і можливості рефакторингу коду.

Перевага використання intelliJ полягає в тому, що

1. Вона швидко генерує getter і setter методи для атрибутів об'єктів.
2. За допомогою простих натискань клавіш ви можете обернути оператор у блоці try-catch або if-else.
3. IDE поставляє вбудовані інструменти для упаковки, такі як gradle, SBT, grunt, bower і т.д.
4. Доступ до баз даних, таких як SQL, ORACLE, PostgreSQL, Microsoft SQL Server, можна отримати безпосередньо з IDE.
5. Він підтримує різні мови, такі як Java, Javascript, Clojure тощо.
6. Він підтримується різними операційними системами, такими як Windows, Linux та ін. Його можна завантажити з офіційного сайту JetBrains.

IntelliJ IDEA була випущена в 2001 році компанією JetBrains, компанією, відомою раніше за свій плагін Resharper для Visual Studio. Community Edition, доступна безкоштовно і орієнтований переважно на Java і розробників додатків Android, пропонує підтримку ряду мов, включаючи Java, Kotlin (розроблений

JetBrains), Groovy, Clojure, Scala та інші. IntelliJ IDEA включає такі функції, як розширене прогнозування, аналіз коду та інтелектуальне завершення коду, а також набір плагінів та розширень для налаштування середовища розробки відповідно до ваших потреб. Плагіни можна завантажувати та встановлювати з сайту репозиторію плагінів IntelliJ або через вбудовану функцію пошуку та встановлення плагінів IDE. В даний час у випуску спільноти IntelliJ IDEA доступно 1495 плагінів, а у версії Ultimate - 1626. Ці цифри набагато менше, ніж у редакторів, таких як Atom, який має понад 7000 пакетів (по суті плагінів). Тим не менш, деякі функції, включені за замовчуванням в IntelliJ, можуть бути додані лише до Atom шляхом встановлення пакунків. Наприклад, linting вбудований в IntelliJ, і він може бути встановлений в Atom, на мові програмування, з різними пакетами.

3.3 Опис інструментів розробки

Для розробки модулю були використані Spring framework (Spring Boot, Spring Security, Spring MVC), PostgreSQL, Maven та засоби для аутентифікації OAuth2 від Google.

3.3.1 **Фреймворк Spring**

Spring Framework - це прикладний фреймворк з контейнером інверсії контролю для платформи Java. Основні можливості можуть використовуватися будь-якими програмами Java, але є розширення для створення веб-додатків поверх платформи Java EE (Enterprise Edition)[4]. Хоча базові елементи не нав'язують жодної конкретної моделі програмування, вона стала популярною в спільноті Java як доповнення до або навіть заміни моделі Enterprise JavaBeans (EJB)[11]. Spring Framework включає кілька модулів, які надають широкий спектр можливостей:

1. Spring Core Container (це базовий модуль Spring, що забезпечує контейнери BeanFactory і ApplicationContext).

2. Аспектно-орієнтоване програмування.

3. Аутентифікація та авторизація: налаштовані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик через підпроект Spring Security.

4. Convention over configuration (швидке рішення для розробки прикладних програм на основі весни пропонується в модулі Spring Roo).

5. Доступ до даних: робота з реляційними системами управління базами даних на платформі Java за допомогою Java Database Connectivity (JDBC) та об'єктно-реляційних засобів відображення та з базами даних NoSQL.

6. Inversion of control container (конфігурація компонентів програми та управління життєвим циклом об'єктів Java, що виконується в основному за допомогою запровадження залежностей).

7. Повідомлення: конфігураційна реєстрація об'єктів слухача повідомлень для прозорого споживання повідомлень з черг повідомлень через Java Message Service (JMS), поліпшення надсилання повідомлень через стандартні JMS API.

8. Model–view–controller (HTTP- і сервлет-основи, що забезпечують гачки для розширення і налаштування для веб-додатків і RESTful (репрезентативна державна передача) веб-служб).

9. Структура віддаленого доступу: конфігураційний віддалений виклик процедур (RPC), сортування Java-об'єктів над мережами, що підтримують виклик віддалених методів Java (RMI), CORBA і протоколи на основі HTTP, включаючи веб-служби SOAP.

10. Управління транзакціями: об'єднує кілька API управління транзакціями і координує транзакції для об'єктів Java.

11. Віддалене керування: конфігураційна експозиція та керування об'єктами Java для локальної або віддаленої конфігурації за допомогою Java Management Extensions (JMX).

12. Тестування: підтримка класів для написання модульних тестів та інтеграційних тестів.

3.3.2 **Фреймворк Spring Boot**

Більш вузькоспеціальною є технологія Spring Boot. Spring Boot - це приклад програми з використанням фреймворку Spring, яку можна "просто запустити". Вона попередньо налаштована з найкращої конфігурації та використання платформи Spring та сторонніх бібліотек, щоб було можливо розпочати роботу з мінімальною суєтою. Більшість програм Spring Boot потребують дуже мало налаштувань Spring.

Особливості:

1. Можна створювати окремі програми Spring.
2. Вбудовувати Tomcat або Jetty безпосередньо (не потрібно розгортати файли WAR).
3. Спрощена конфігурація Maven.
4. Автоматично можна налаштувати деталі Spring, коли це необхідно.
5. Абсолютно ніякої генерації коду та вимоги до конфігурації XML[10].

3.3.3 **Фреймворк Spring Security**

Spring Security - фреймворк, що легко настроюється та є дуже потужним. Це фактичний стандарт для забезпечення додатків на основі Spring.

Spring Security - це система, яка фокусується на забезпеченні аутентифікації та авторизації до Java-додатків. Як і всі Spring-проекти, реальна сила Spring Security

виявляється в тому, наскільки легко його можна розширити, щоб задовольнити вимоги користувача[5].

Особливості:

1. Комплексна та розширювана підтримка як для аутентифікації, так і для авторизації.
2. Захист від атак, таких як фіксація сеансів, підробка запитів на сайтах і т.д.
3. Інтеграція API сервлетів.
4. Додаткова інтеграція з Spring Web MVC.

3.3.4 **Фреймворк Spring Web MVC**

Фреймворк Spring Web MVC надає архітектуру Model-View-Controller (MVC) і готові компоненти, які можуть бути використані для розробки гнучких і вільно пов'язаних веб-додатків[12]. Шаблон MVC призводить до розділення різних аспектів програми (вхідна логіка, бізнес-логіка і логіка UI), забезпечуючи при цьому вільну зв'язок між цими елементами.

1. Модель інкапсулює дані додатків і в цілому вони складаються з POJO(звичайний класс Java).
2. View відповідає за візуалізацію даних моделі і в цілому генерує на вихід HTML, який інтерпретує браузер клієнта.
3. Контролер відповідає за обробку запитів користувачів і створення відповідної моделі і передає їх у View для рендеринга.

Нижче наведено послідовність подій, що відповідають вхідному HTTP-запиту до DispatcherServlet :

1. Після отримання HTTP-запиту DispatcherServlet консультує HandlerMapping для виклику відповідного контролера.

2. Контролер приймає запит і викликає відповідні методи обслуговування на основі використовуваного методу GET або POST. Метод служби встановлює дані моделі на основі визначеної бізнес-логіки і повертає ім'я view в DispatcherServlet .

3. DispatcherServlet допоможе ViewResolver підібрати визначений view запиту.

4. Після завершення перегляду, DispatcherServlet передає дані моделі view, який, нарешті, відтворюється у браузері.

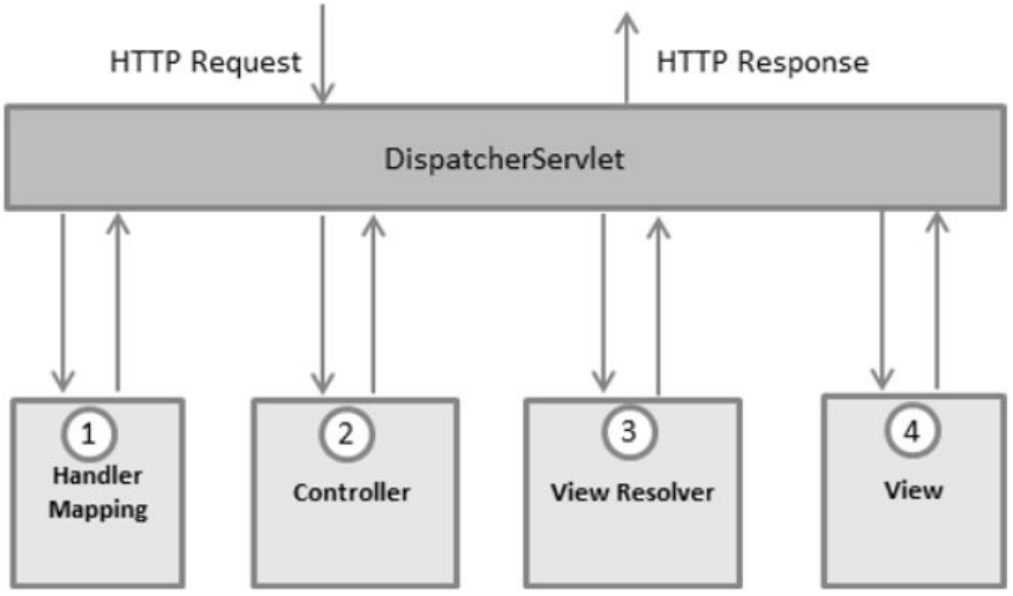


Рисунок 3.1 – Процес обробки запитів за допомогою DispatcherServlet

Структура Spring Web-модель-контролер (MVC) створена навколо DispatcherServlet, який обробляє всі HTTP-запити і відповіді. Робочий процес обробки запитів Spring Web MVC DispatcherServlet ілюструється на наступній схемі.

Всі вищезгадані компоненти, тобто HandlerMapping, Controller і ViewResolver є частинами WebApplicationContext, які є розширенням plainApplicationContext з деякими додатковими функціями, необхідними для веб-додатків.

3.3.5 **Технологія Maven**

Maven - це популярний інструмент для створення корпоративних Java-проектів з відкритим вихідним кодом, призначений для більшої роботи з процесу складання. Maven використовує декларативний підхід, де описані структура і зміст проекту, а не підхід на основі завдань, що використовується в Ant або в традиційних файлах, наприклад. Це допомагає забезпечити дотримання стандартів розвитку всієї компанії і скорочує час, необхідний для запису і збереження сценаріїв збірки.

Основою проекту Maven 2 є об'єктна модель проекту (або коротко POM). Він містить докладний опис проекту, включаючи інформацію про керування версіями та конфігурацією, залежності, ресурси програми та тестування, членів команди та структуру та багато іншого. POM приймає форму XML-файлу (pom.xml), який розміщується у домашньому каталозі проекту[13].

Значна частина переваг Maven походить від стандартної практики, яку він робить. Розробник, який раніше працював над проектом Maven, відразу відчує структуру та організацію нового. Не потрібно витрачати даремно час на організацію структур каталогів, умов і налаштованих сценаріїв побудови для кожного проекту. Незважаючи на те, що можна перевизначити будь-яке місце розташування каталогу для власних конкретних цілей, необхідно дотримуватись стандартної структури каталогів Maven з кількох причин:

1. Це робить файл POM меншим і простішим.
2. Це робить проект легшим для розуміння і полегшує роботу іншим розробникам.
3. Це полегшує інтеграцію плагінів.

Життєвий цикл проекту займає центральне місце в Maven 2. Більшість розробників знайомі з поняттями етапів побудови, таких як компіляція, тестування і розгортання. Ant має цілі з такими іменами. У Maven 1 відповідні плагіни викликаються безпосередньо. Наприклад, для компіляції вихідного коду Java використовується модуль Java: $maven java:compile. У Maven 2 це поняття стандартизовано у набір добре відомих та чітко визначених фаз життєвого циклу. Замість того, щоб викликати плагіни, розробник Maven 2 викликає фазу життєвого циклу: $mvn compile.

Деякі з найбільш корисних фаз життєвого циклу Maven 2 такі:

gener-source: генерує будь-який додатковий вихідний код, необхідний для програми, який, як правило, виконується за допомогою відповідних плагінів

1. compile: компілює вихідний код проекту.
2. test-compile: компілює тестові одиниці проекту.
3. test: Запускає модульні тести (як правило, використовуючи JUnit) у каталозі src / test.
4. package: упаковує скомпільований код у форматі, який можна розповсюджувати (JAR, WAR і т.д.)
5. integration-test: обробляє і розгортає пакет, якщо необхідно, в середовище, в якому можуть виконуватися інтеграційні тести.
6. install: Встановлює пакет у локальне сховище для використання як залежність в інших проектах на вашому локальному комп'ютері.
7. deploy: завершується в середовищі інтеграції або випуску, копіює кінцевий пакет у віддалене сховище для обміну з іншими розробниками і проектами.

Ці етапи ілюструють переваги рекомендованої практики, яку рекомендує Maven 2: як тільки розробник ознайомиться з основними фазами життєвого циклу

Maven 2, він повинен відчувати себе невимушено у фазах життєвого циклу будь-якого проекту Maven.

Фаза життєвого циклу викликає плагіни, необхідні для виконання завдання. Виклик фази життєвого циклу автоматично викликає будь-які попередні фази життєвого циклу. Оскільки фази життєвого циклу обмежені за кількістю, їх легко зрозуміти і добре організувати, ознайомлення з життєвим циклом нового проекту Maven 2 проходить дуже просто.

Одним з основних моментів Maven 2 є транзитивне управління залежністю. З Maven 1, необхідно оголосити кожну JAR, яка буде потрібна, прямо чи опосередковано, програмою. З Maven 2 не потрібно піклуватися за відповідні JAR файли для програмного модулю. Необхідно лише надати Maven інформацію, які бібліотеки потрібні, і Maven буде піклуватися про бібліотеки, які потрібні модулям.

У реальному корпоративному додатку, можливо, не потрібно буде включати всі залежності в розгорнутому додатку. Деякі JAR-файли потрібні лише для модульного тестування, тоді як інші будуть надаватися під час виконання сервером додатків. Використовуючи техніку, що називається визначенням залежності, Maven 2 дозволяє використовувати деякі JAR тільки тоді, коли вони дійсно потрібні, і виключає їх з classpath, коли цього не відбувається.

Maven надає чотири області залежності:

1. compile: Залежність обсягу компіляції доступна на всіх етапах. Це значення за замовчуванням.

2. provided: Вказана залежність використовується для компіляції програми, але вона не буде розгорнута. Необхідно використовувати цю область, якщо очікується, що JDK або сервер додатків надасть JAR. API сервлетів є гарним прикладом.

3. runtime: Залежність Runtime-scope не потрібна для компіляції, тільки для виконання, наприклад, драйверів JDBC (Java Database Connectivity).

4. test: Залежність тестової області потрібна лише для компіляції та запуску тестів (наприклад, JUnit).

З мінімальними зусиллями, можна мати професійний веб-сайт з якісним проектом.

Розробка та підтримка веб-застосунку стає набагато легшою, коли Maven інтегрований в процес збирання сайта за допомогою безперервної інтеграції або навіть автоматичних нічних збірок. Типовий сайт Maven може щодня публікувати:

1. Загальну інформація про проект, наприклад, сховища джерел, відстеження дефектів, члени команди і т.д.
2. Звіти про тестові та тестові покриття.
3. Автоматичні огляди кодів і з Checkstyle і PMD.
4. Інформація про конфігурацію та версії.
5. Залежності.
6. Javadoc
7. Вихідний код в індексованому та перехресному форматі HTML.
8. Список членів команди.

Maven 2.0 є потужним інструментом, що значно спрощує і стандартизує процес збирання. Просуваючи стандартну організацію проекту та рекомендовані дії, Maven бере на себе більшу частину роботи. А стандартні плагіни, такі як генератор сайтів, забезпечують цінні інструменти проекту, які не потребують додаткових зусиль.

3.3.6 **База даних PostgreSQL**

При написанні даних програмних засобів була використана база даних PostgreSQL.

PostgreSQL, також відома як просто Postgres, є вільною і відкритою системою управління реляційними базами даних (СУБД), що підкреслює розширюваність і відповідність стандартам. Він призначений для обробки робочих навантажень від окремих машинних додатків до сховищ даних або веб-служб з багатьма одночасними користувачами. Це база даних за замовчуванням для MacOS Server і доступна також для Linux, FreeBSD, OpenBSD і Windows. Однією з особливостей даної бази даних є Multiversion concurrency control[6].

PostgreSQL керує паралельністю через багатоваріантне управління паралелізмом (MVCC), яке надає кожній транзакції "знімок" бази даних, дозволяючи вносити зміни без впливу на інші транзакції. Це значною мірою позбавляє потреби в блокуваннях читання і гарантує, що база даних підтримує принципи ACID[14]. PostgreSQL пропонує три рівні ізоляції транзакцій: Read Committed, Repeatable Read і Serializable. Тому що PostgreSQL не застрахований від брудних читань, запитуючи рівень ізоляції транзакцій Read Uncommitted забезпечує замість цього повідомлення. PostgreSQL підтримує повну серійну здатність за допомогою методу SSI.

PostgreSQL поставляється з багатьма функціями, які допомагають розробникам створювати додатки, адміністраторам - захистити цілісність даних, а також допомагати керувати вашими даними незалежно від того, наскільки великий або малий набір даних. На додаток до того, що PostgreSQL є вільним і відкритим кодом, він дуже розширюється. Наприклад, ви можете визначити власні типи даних, побудувати власні функції, навіть написати код з різних мов програмування без перекомпіляції бази даних.

3.3.7 **Технологія аутентифікації OAuth2**

Для надання доступу користувачеві до сервісу, було прийняте рішення використати OAuth2 від Google. OAuth є відкритим стандартом для делегування

доступу, що зазвичай використовується як спосіб для користувачів Інтернету надавати веб-сайтам або програмам доступ до їхньої інформації на інших веб-сайтах, але без надання їм паролів. Цей механізм використовують такі компанії, як Amazon, Google, Facebook, Microsoft і Twitter, щоб дозволити користувачам обмінюватися інформацією про свої облікові записи з додатками або веб-сайтами третіх сторін.

OAuth - це послуга, яка доповнює OpenID і відрізняється від неї. OAuth також відрізняється від OATH, яка є еталонною архітектурою для аутентифікації, а не стандартом для авторизації. Проте OAuth безпосередньо пов'язаний з OpenID Connect (OIDC), оскільки OIDC є шаром аутентифікації, побудованим на вершині OAuth 2.0. OAuth також відрізняється від XACML, який є стандартом політики авторизації. OAuth можна використовувати в поєднанні з XACML, де OAuth використовується для отримання згоди та доступу до даних, тоді як XACML використовується для визначення політики авторизації.

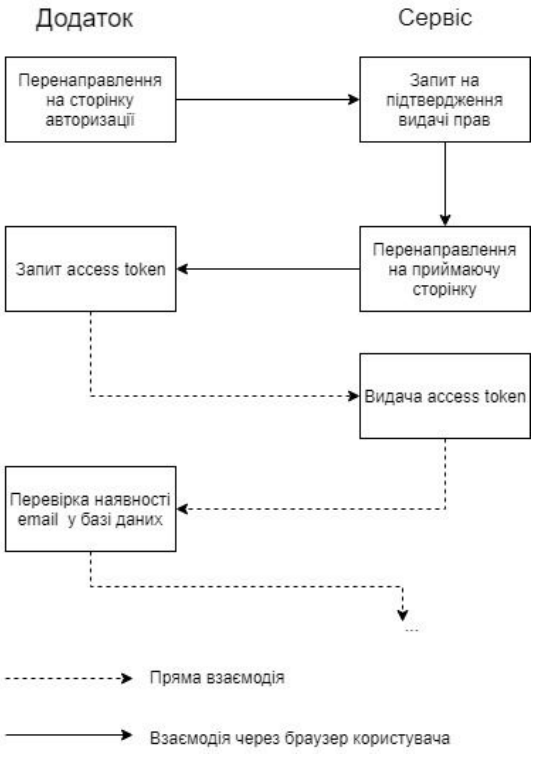


Рисунок 3.2 – Архітектура авторизації за допомогою OAuth2

Як правило, OAuth надає клієнтам "безпечний делегований доступ" до ресурсів сервера від імені власника ресурсу. Вона визначає процес для власників ресурсів, щоб дозволити доступ третіх сторін до своїх ресурсів сервера без обміну їхніми обліковими даними. Розроблений спеціально для роботи з протоколом передачі гіпертексту (HTTP), OAuth, по суті, дозволяє запросити токени доступу третій стороні сервером авторизації з дозволу власника ресурсу. Потім третя сторона використовує токен доступу для доступу до захищених ресурсів, розміщених на сервері ресурсів.

3.3.8 **Рекомендації щодо впровадження G Suite**

G Suite - раніше відомий як Google Apps for Work - продукт із програмним забезпеченням (SaaS), який групує всі хмарні продуктивність і засоби співпраці, розроблені Google для підприємств, інститутів і неприбуткових організацій. Ви отримуєте доступ до власних адрес Gmail, Документів, Таблиць, Слайдів, Календарів, Диску, Сайтів та багато іншого, що є включеними до кожної підписки. Рекомендується поєднувати ресурс, що використовує OAuth2 з G Suite для організації роботи з даними, що знаходяться на цьому ресурсі. Тобто за допомогою G Suite є можливість створити групу акаунтів Гугл з бажаним доменом та використовувати ці акаунти лише для корпоративних цілей.

Висновки до розділу

Для розробки веб-додатку хмарного сховища було прийнято рішення використовувати мову програмування Java та набір фреймворків, таких як Spring, Spring Boot, Spring Security та Spring MVC. Для контрольованого доступу до хмарного сховища прийнято рішення використовувати OAuth2. Дана методика

дозволяє скористатись усією потужністю захисту персональних даних та доступу до хмарно сховища засобами Google. А дані про існуючі права доступу всередині хмарного сховища зберігати у базі даних PostgreSQL.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Серверна частина проекту виглядає таким чином:

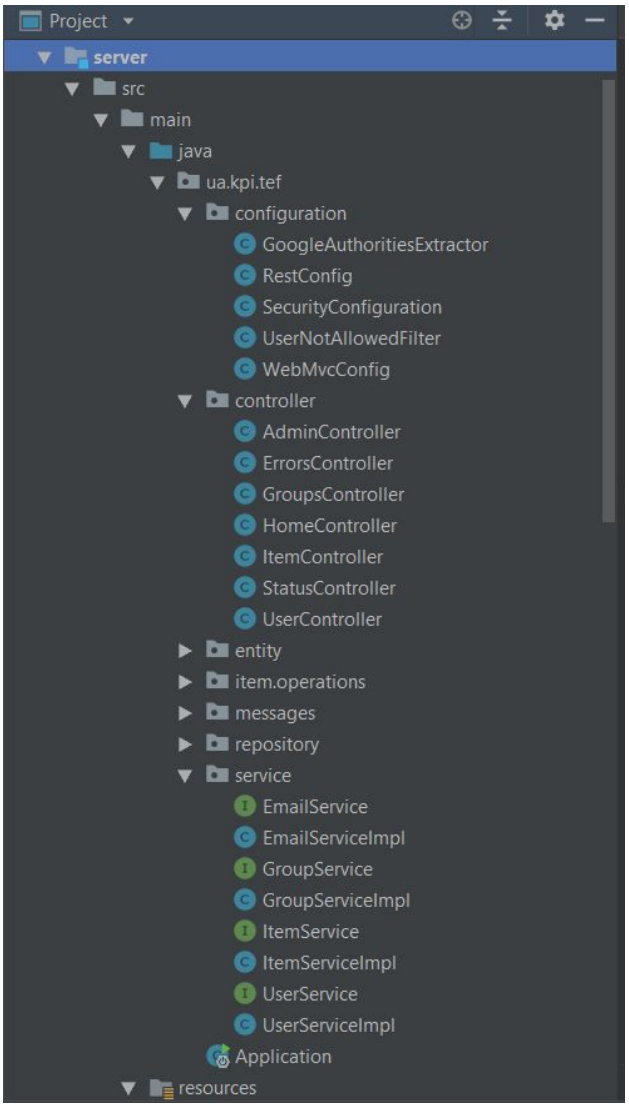


Рисунок 4.1 – Ієрархія директорій та файлів проекту

Основними директоріями є configuration, controller та service.

У директорії service знаходяться класи для роботи веб-застосунку за базою даних. Дані класи спрямовані на прийом файлів, їх обробку та передачу. Для роботи з кожною окремою сутністю було розроблено окремий клас відповідно до вимог чистого коду.

У папці controller знаходяться класи що містять endpoints, які є основними засобами при роботі на серверній частині веб-додатку. за допомогою цих методів виконується прийом даних, перевірки та передача даних у сервіси для подальшої обробки.

У папці configuration знаходяться класи, які є основними у конфігурації веб-застосунку. Саме ці класи відповідають за організацію контрольованого доступу до хмарного сховища та за допомогою цих класів можна дізнатися всю інформацію про користувача, який долучається до системи.

4.1 Функціональність модулю контролю доступу

Даний модуль направлений на організацію контрольованого доступу до хмарного сховища. Даний механізм виконується за допомогою OAuth2. Якщо даний користувач є у базі даних користувачів - він допускається до користування даним веб-додатком. Не менш важливим аспектом є контрольований доступ всередині хмарного сховища. Користувач повинен переглядати та оперувати лише над тими файлами, які йому доступні. Це досягається за допомогою присвоєння певних рівнів доступу до певних файлів. Відповідно до рівня доступу вирішується подальший сценарій дій користувача над файлом.

До системи хмарного сховища може запросити лише Адміністратор сховища. Після запрошення, користувачу надсилається на електронну пошту лист про те, що йому доступна можливість використовувати дане хмарне сховище. Адміністратор залишає за собою право вибору рівня доступу даного користувача по системі, а також має можливість видалити користувача із системи.

Для полегшення оперування великими кількостями користувачів, було запроваджено групи користувачів. Даний функционал надає можливість проводити операції відразу над декількома користувачами. Користувач може бути запрошений

до групи, а може створити свою. також він має право в односторонньому порядку вийти з групи, до якої він був доданий.

Користувач може завантажувати свої файли до сховища та виставляти відповідні рівні доступу відповідним користувачам за власним бажанням.

4.2 Структура таблиць бази даних

База даних даної системи хмарного сховища складається з таблиць, які слугують для збереження інформації про користувачів, групи користувачів, файли та рівні доступу користувачів у системі та до файлів.

Першою таблицею є таблиця account, що зберігає дані про користувача системи і має таку структуру:

	email	user_id	role	user_name
1	jack.nesterko@gmail.com	1	ROLE_ADMIN	Eugene Nesterko
2	eugene.kraros@gmail.com	112438065519870960914	ROLE_ADMIN	Евгений Ковтун

Рисунок 4.2 – Структура таблиці account

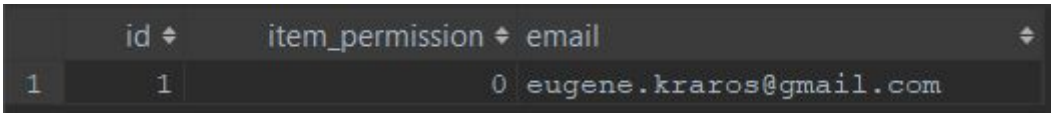
Друга таблиця, що називається grp, є відповідальною за групи користувачів и містить лише ідентифікатор та назву групи. Третя таблиця account_to_group відповідає співвідношенню користувача до групи в якій він знаходиться.

Четверта таблиця відповідає за збереження інформації про файл, що було завантажено у хмарне сховище, називається item і має таку структуру:

	item_id	is_directory	last_modified	name	path	size	parent_item_id
1	1	☑	1559859345587	12334	./root/12334	<null>	<null>
2	2	☐	1559859400686	7.jpeg	./root/7.jpeg	21514	<null>
3	3	☐	1559859400688	3.jpeg	./root/3.jpeg	30536	<null>
4	5	☐	1559859400686	4.jpeg	./root/4.jpeg	11080	<null>
5	4	☐	1559859400688	8.jpeg	./root/8.jpeg	17736	<null>
6	6	☐	1559859400700	2.jpeg	./root/2.jpeg	47235	<null>
7	7	☐	1559859400687	6.jpeg	./root/6.jpeg	16047	<null>
8	8	☐	1559859400971	1.jpeg	./root/1.jpeg	30139	<null>

Рисунок 4.3 – Структура таблиці item

П’ята таблиця user_to_item відповідає за співвідношення користувача з файлом, та його доступом до цього файлу і має наступну структуру:



	id	item_permission	email
1	1	0	eugene.kraros@gmail.com

Рисунок 4.4 – Структура таблиці user_to_item

Висновки до розділу

У даному розділі було описано основну функціональність програмного модулю контролю доступу до хмарного сховища та контролю доступу до файлів всередині хмарного середовища між користувачами. Також були описані можливості адміністратора у системі. Дані для роботи програмного модуля зберігаються у базі даних PostgreSQL.

5 МЕТОДИКА РОБОТИ КОНТРОЛЬОВАНОГО ДОСТУПУ ДО ХМАРНОГО СХОВИЩА

Даний модуль розроблений з використанням веб-технологій, отже працюватиме у всіх сучасних браузерах.

5.1 Системні вимоги та інсталяція

Дана система було повністю впроваджено та розгорнута за допомогою Heroku. Тобто користувачу нічого не потрібно крім сучасного браузера та доступу до Інтернету. Користувачу лише необхідно перейти за посиланням https://star-file-system.herokuapp.com/.

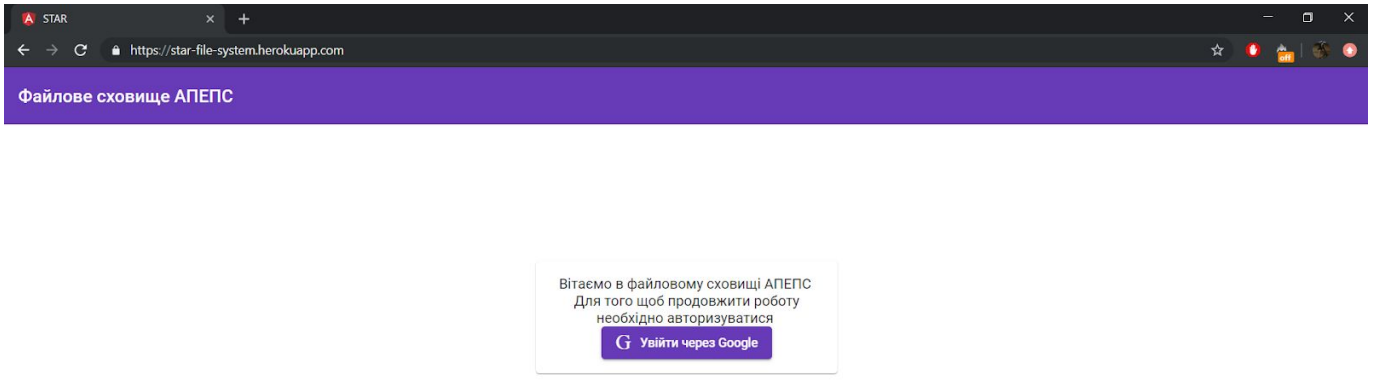


Рисунок 5.1 – Початок роботи із веб-додатком після переходу за посиланням

5.2 Інструкція з використання контрольованого доступу

При вході у веб-додаток, клієнту перш за все потрібно авторизуватися натиснувши на відповідну кнопку. Після цього користувачу буде запропоновано авторизуватися за допомогою свого акаунту Google, що має свій шар безпеки. Це значно полегшує контроль доступу до файлового сховища, та робить авторизацію більш зручною для самого користувача – йому не потрібно завжди пам’ятати логін і пароль від даного веб-додатку, якщо він пам’ятає свої дані для авторизації у Google.

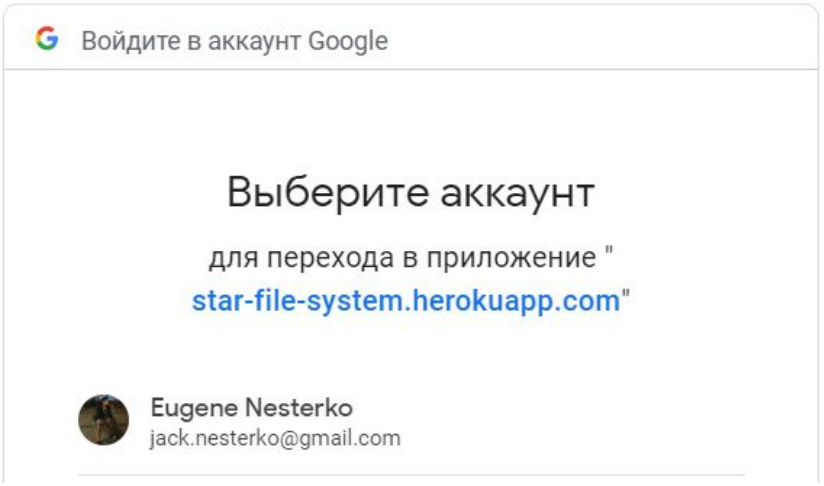


Рисунок 5.2 – Акаунти на вибір для авторизації за допомогою OAuth2

Після цього користувач потрапляє до головного меню веб-додатку і має можливість завантажувати та оперувати своїми файлами, та файлами, що йому доступні.

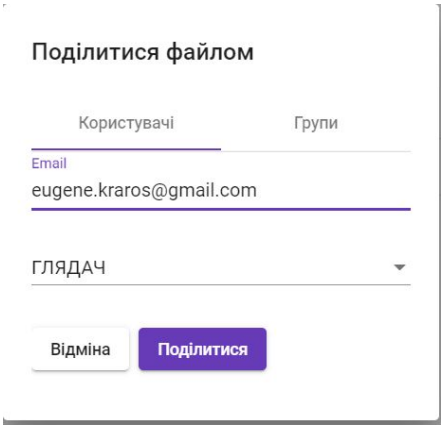


Рисунок 5.3 – Модальне вікно для функції “Поділитися”

Користувач може поділитися своїм завантаженим файлом з користувачем або з цілою групою користувачів.

Також користувач може переглянути кому саме доступний його файл, або папка через функцію “Контроль доступу”.

Дану функцію можна викликати за допомогою правого кліку мишки на файл або папку.

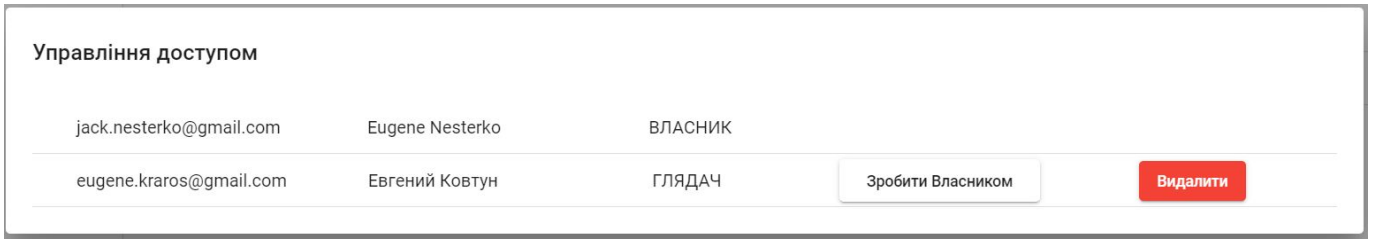


Рисунок 5.5 – Модальне вікно для функції “Контроль доступу”

Користувачі можуть об’єднуватись у групи, та оперувати своїми групами в залежності від рівня доступу у цій групі. Для цього необхідно перейти у розділ “Групи” навігаційного меню.

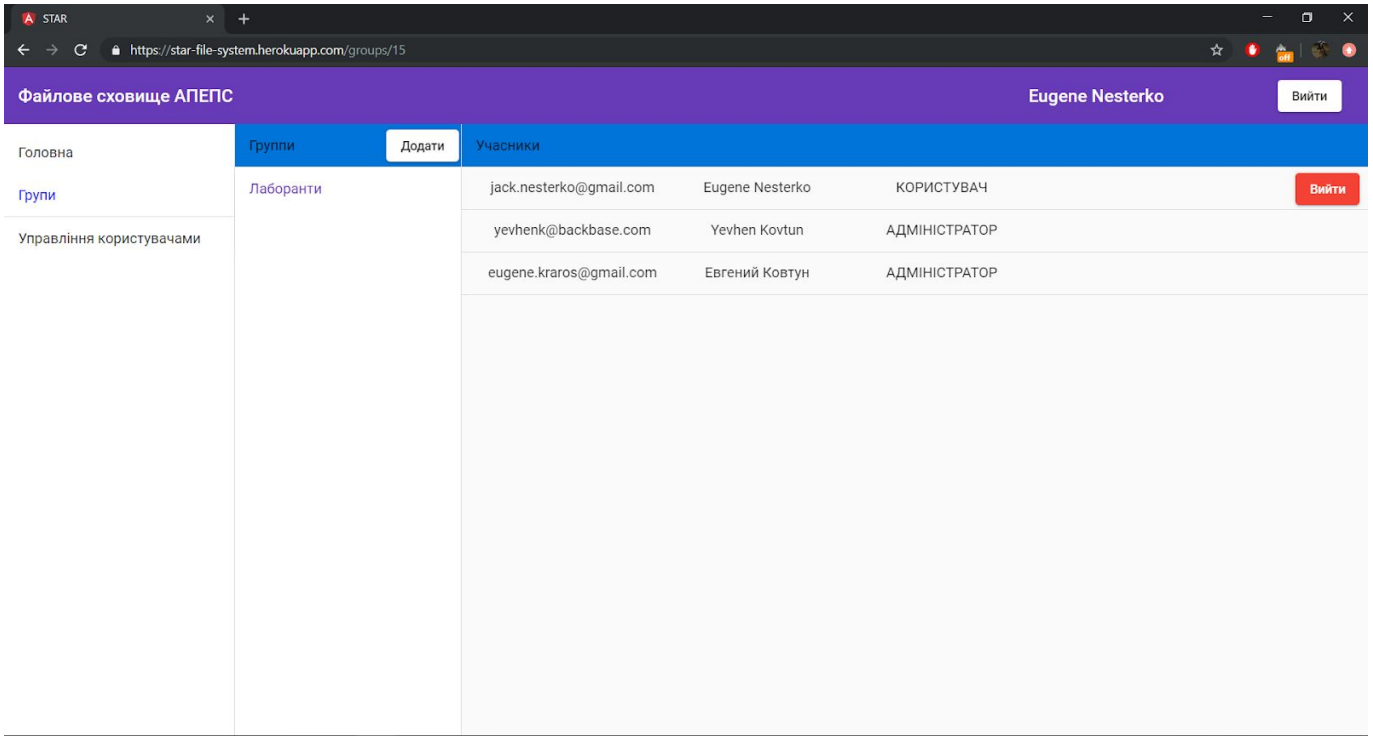


Рисунок 5.6 – Інтерфейс вкладки “Групи” навігаційного меню

Користувач може створювати та видаляти нові групи, а також - запрошувати існуючих користувачів до групи. Якщо сам користувач вже був доданий до певної групи, то він може вийти з неї.

Адміністратор системи може так само користуватися всім основним функціоналом веб-додатку и також може керувати користувачами системи.

При переході до розділу “Управління користувачами”, адміністратор може побачити існуючих користувачів системи та управляти їми.

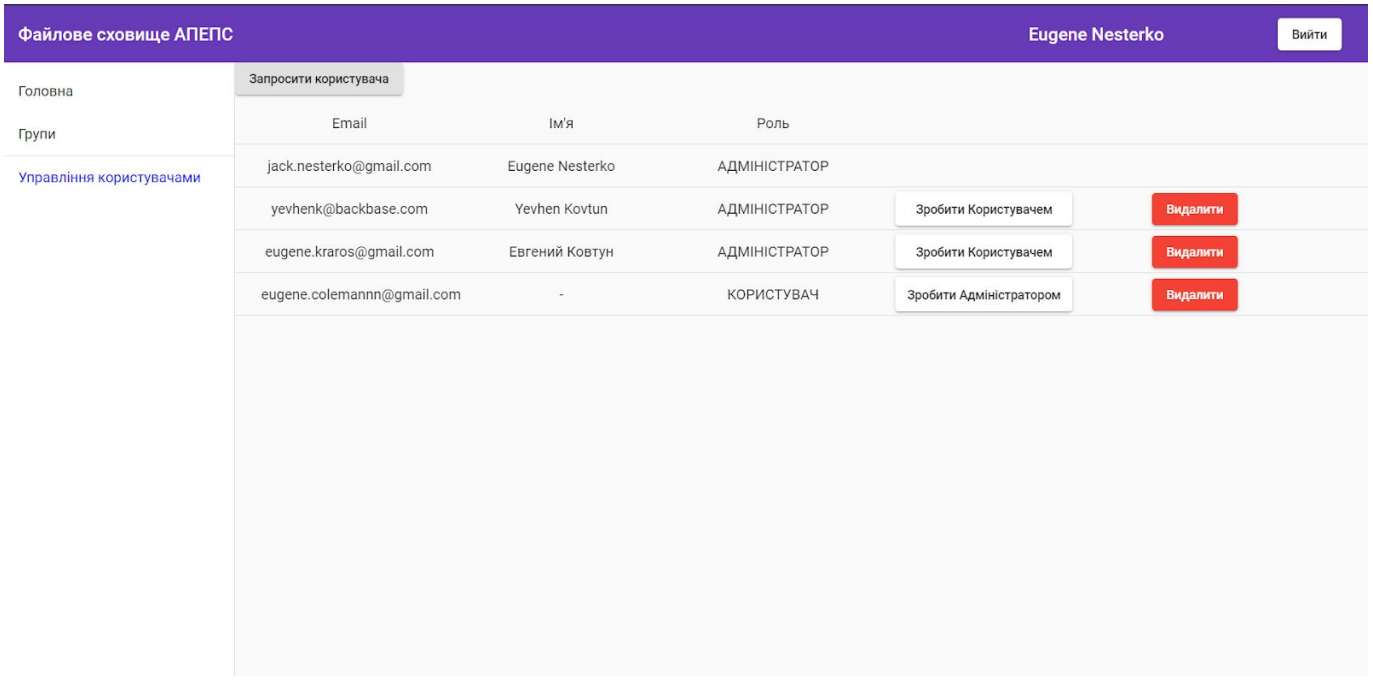


Рисунок 5.7 – Інтерфейс вкладки “Управління користувачами” навігаційного меню

Адміністратор може додати до системи нового користувача кліком на кнопку запросити користувача.

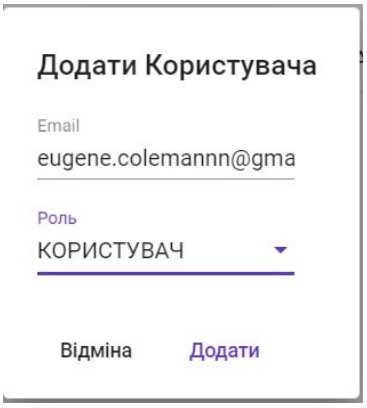


Рисунок 5.8 – Модальне вікно для функції “Запросити користувача”

Потрібно ввести електронну пошту даного користувача, та його майбутній доступ по системі.

Після цього даному користувачу буде відправлено лист на електронну пошту про те, що його було додано користувачем до кафедрального хмарного сховища.

У даному листі знаходиться посилання, перйшовши по якому користувач матиме змогу розпочати роботу з веб-додатком.

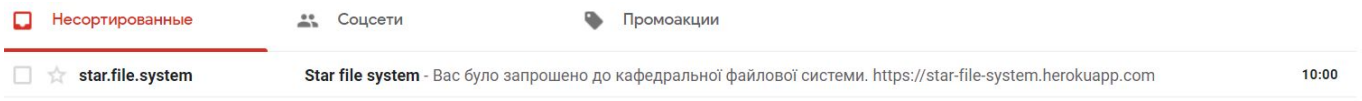


Рисунок 5.9 – Електронний лист, що сповіщає користувача

Висновки до розділу

У цьому розділі було описано способи користування модулем контролю доступу до хмарного сховища. Специфічні вимоги до системи або до інсталяції відсутні, оскільки хмарне сховище реалізовано як веб-додаток і потребує лише веб-браузера та стабільного з’єднання з інтернетом.

ВИСНОВКИ

При детальному вивченні проблематики заданої теми дипломної роботи, були проаналізовані рішення серед сучасних технологій. Таким чином було запроваджено засоби для надійної та комфортної авторизації користувача за допомогою сервісів Google та технології OAuth2. При авторизації, користувачу не потрібно вводити додаткові дані, достатньо авторизації у свій Google-акаунт. Додатково виконується перевірка за email-адресою, щоб розуміти: чи надавати доступ даному користувачеві до хмарного сховища та до яких саме файлів надати доступ певного рівня. Користувачів було розділено на дві групи: звичайні користувачі та адміністратори. Усі користувачі мають змогу бути згрупованими по групах. При запрошенні до системи, користувачу відправляється лист на електронну пошту з посиланням, по якому необхідно перейти для початку роботи з веб-додатком.

Всі ці засоби були зібрані в програмні модулі за допомогою мови Java та фреймворків Spring Boot, Spring Security та Spring MVC. Дані засоби є кросплатформеними, тобто можуть функціонувати на пристроях будь-яких платформ.

Було виявлено переваги користування фреймворком для авторизації OAuth2, оскільки недоліки методу авторизації, що є базовим у Spring Security, повністю усунені у фреймворку OAuth2. Також даний фреймворк надає системі велику кількість інформації про користувача, який нею користується.

Усі дані про користувачів, їх рівні доступу, про файли, та рівні доступу файлів було зібрано у базу даних PostgreSQL.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Петров В.В., Нестеренко О.В., Монастирецький М.Г., Шагалов В.Ю. Національні інформаційні ресурси. Проблеми формування, розвитку, управління і використання // Реєстрація, зберігання і оброб. даних. — 2001. — Т. 3, № 2. — С. 38–49.

2. Дубова Н. SOA: подходы к реализации // Открытые системы. — 2004. — № 6. — С. 37–41.

3. Ньюкомер Э. Web-сервисы. Для профессионалов. — СПб.: Питер, 2003. — 256 с.

4. Кларенс Хо, Роб Харроп. Spring 3 для профессионалов. — М. : «Вильямс», 2012. — 880 с. — ISBN 978-5-8459-1803-1.

5. Крейг Уоллс. Spring в действии. — Третє. — М. : «Manning», 2014. — 624 с. — ISBN 9781617291203.

6. Завадський І. Основи баз даних / Ігор Завадський. – Київ: ПП І.О. Завадський, 2004. – 192 с.

7. Lamont I. Google Drive & Docs in 30 Minutes (2nd Edition): The unofficial guide to the new Google Drive, Docs, Sheets & Slides / Ian Lamont., 2015. – 112 с.

8. Parecki A. OAuth 2.0 Framework [Електронний ресурс] — Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6749.

9. SpringSource. Spring Security [Електронний ресурс] — Режим доступу до ресурсу: https://spring.io/projects/spring-security.

10. SpringSource. Spring Boot [Електронний ресурс] — Режим доступу до ресурсу: https://spring.io/projects/spring-boot.

11. SpringSource. Spring [Електронний ресурс] — Режим доступу до ресурсу: https://spring.io/projects/spring-framework.

12. SpringSource. Web MVC framework [Електронний ресурс] — Режим доступу до ресурсу: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html.

13. Sonatype. Maven Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://maven.apache.org/guides/.

14. Стоунбрейкер М. PostgreSQL: Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.postgresql.org/docs/

15. Microsoft Azure Documentation | Microsoft Docs [Електронний ресурс] — Режим доступу до ресурсу: https://docs.microsoft.com/en-us/java/azure/?view=azure-java-stable.

16. Google. Cloud Storage documentation [Електронний ресурс] / Google – Режим доступу до ресурсу: https://cloud.google.com/storage/docs/.

17. Amazon. Amazon S3 AWS [Електронний ресурс] / Amazon – Режим доступу до ресурсу: https://docs.aws.amazon.com/AmazonS3/latest/dev/.

18. Richard E. Smith. Authentication: From Passwords to Public Keys / Richard E. Smith., 2001. – с. 231 – (ISBN-13: 978-0201615999).

19. Барсегян А. А., Куприянов М. С., Степаненко В. В., Холод И. И. Методы и модели анализа данных: OLAP и Data Mining. — СПб.: БХВ-Петербург, 2004. — 336 с.

20. Шишкін В.М. Безпека хмарних обчислень – проблеми та можливості ризик-аналізу [Текст] / В.М. Шишкін// Міжнародна наукова конференція “Автоматизовані системи управління та сучасні інформаційні технології”. Тези доповідей – Tbilisi: Publication House “Technical University”, 2011. – С. 142.

21. Електронні сховища даних із захищеним доступом / Є. Б. Артамонов, О. О. Бєляков // Наукоємні технології. - 2013. - № 4. - С. 402-405. - Режим доступу: http://nbuv.gov.ua/UJRN/Nt_2013_4_10 .

22. Inmon W.H. Building the Data Warehouse, 4th Edition. — Hoboken, NJ:Wiley, 2005. — 576 p.

ДОДАТОК А

Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища.

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51157_19Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ ТВ41107_18Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ ТВ41107_18Б 12-1	SecurityConfiguration.java ItemController.java User.java	Основні компоненти
УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ ТВ41107_18Б 13-1	Додаток В.doc	Опис програмного модуля

ДОДАТОК Б

Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища.

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51157_19Б 12-1

Аркушів 15

Київ 2019

```
package ua.kpi.tef.configuration;

import org.springframework.beans.factory.annotation.Qualifier;
import
org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;
import
org.springframework.boot.autoconfigure.security.oauth2.resource.PrincipalExtractor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import org.springframework.security.access.AuthorizationServiceException;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigure
rAdapter;
import org.springframework.security.web.firewall.HttpFirewall;
import org.springframework.security.web.firewall.StrictHttpFirewall;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import ua.kpi.tef.entity.User;
import ua.kpi.tef.repository.UserRepository;

@Configuration
@EnableWebSecurity
@EnableOAuth2Sso
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {




    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http
```

```
                .csrf()
                .disable()
                .authorizeRequests()
                   .antMatchers("/login", "/get_roles","/error_page", "/").permitAll()
                   .antMatchers("/admin").hasAuthority("ROLE_ADMIN")
                   .antMatchers("/admin_api/*").hasAuthority("ROLE_ADMIN")
                                                        .antMatchers("/api/**",   "/main*",
"/groups_api/*","/user_api/*").hasAnyAuthority("ROLE_ADMIN", "ROLE_USER")
                   .anyRequest()
                   .authenticated()
                .and()
                   .formLogin()
                   .loginPage("/login")
                   .loginProcessingUrl("/perform_login")
                                .successHandler((request, response, authentication) ->
response.setStatus(200))
                                    .failureHandler((request, response, exception) ->
response.setStatus(401))
                   .usernameParameter("userName")
                   .passwordParameter("password")
                .and()
                   .logout()
                   .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                   .logoutSuccessUrl("/login");



        }

        @Override
        public void configure(WebSecurity web) {
           web
                .httpFirewall(allowUrlEncodedSlashHttpFirewall())
                .ignoring()
                //  .antMatchers("/resources/**");
                     .antMatchers("/resources/**", "/runtime.*", "/main.*", "/polyfills.*",
"/styles.*", "/favicon.ico");
```

```
    }

    @Bean
    public HttpFirewall allowUrlEncodedSlashHttpFirewall() {
       StrictHttpFirewall firewall = new StrictHttpFirewall();
       firewall.setAllowUrlEncodedSlash(true);
       return firewall;
    }



    @Bean
        public PrincipalExtractor principalExtractor(@Qualifier("userRepository")
UserRepository userRepository) {
        return map -> {
           String email = (String) map.get("email");
          User user =  userRepository.findByEmail(email).map((newUser) -> {
             newUser.setId((String) map.get("sub"))
                   .setUserName((String) map.get("name"))
                   .setEmail((String) map.get("email"));
             System.out.println(newUser);
             return newUser;
           }).orElseThrow(
                                                                  () -> new
AuthorizationServiceException(HttpStatus.FORBIDDEN.toString())
           );
           return userRepository.save(user);

        };
    }



}
package ua.kpi.tef.controller;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import ua.kpi.tef.entity.*;
import ua.kpi.tef.item.operations.ItemOperations;
import ua.kpi.tef.service.GroupService;
import ua.kpi.tef.service.ItemService;
import ua.kpi.tef.service.UserService;

import javax.xml.ws.http.HTTPException;
import java.util.*;

@RestController
@RequestMapping("/api")
public class ItemController {

   private final GroupService groupService;
   private final UserService userService;
   private final ItemService itemService;

   @Autowired
      public ItemController(GroupService groupService, UserService userService,
ItemService itemService) {
      this.groupService = groupService;
      this.userService = userService;
      this.itemService = itemService;
   }

   @GetMapping("/get/{id}")
    Item getItemById(@PathVariable Long id, @AuthenticationPrincipal User user)
{
```

```
                                                                                    if
(userService.findUserByEmail(user.getEmail()).get().getItems().containsKey(itemService.
findById(id).get())) {
                    return itemService.findById(id).orElse(null);
                } else {
                    throw new HTTPException(403);
                }

            }

            @GetMapping(value = "/getAllItems")
            @ResponseBody
             public List<ItemWithPermission> getAllItems(@RequestParam(required = false)
Long idParent, @AuthenticationPrincipal User user) {
                System.out.println("In Get All Items");
                User currentUser = userService.findUserByEmail(user.getEmail()).get();

                if (currentUser.getItems() == null) {
                    return new ArrayList<>();
                }

                Map<Item, ItemPermission> items = currentUser.getItems();
                List<ItemWithPermission> itemWithPermissions = new ArrayList<>();
                                    Item parentItem = idParent == null ? null :
itemService.findById(idParent).get();
            //      Item parentItem = itemService.findById(idParent).orElse(null);
                if (parentItem == null) {
                    for (Map.Entry<Item, ItemPermission> entry : items.entrySet()) {
                        if (entry.getKey().getParent() == null) {
                                itemWithPermissions.add(new ItemWithPermission(entry.getKey(),
entry.getValue()));
                        }
                    }
                } else {
                    for (Map.Entry<Item, ItemPermission> entry : items.entrySet()) {
                        if (parentItem.equals(entry.getKey().getParent())) {
```

```
                ItemWithPermission itemWithPermission = new
ItemWithPermission(entry.getKey(), entry.getValue());
                    itemWithPermissions.add(itemWithPermission);
                }
            }
        }
        return itemWithPermissions;
    }

                @PostMapping(value = "/upload", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    @ResponseBody
     public ResponseEntity<String> uploadItem(@RequestParam MultipartFile file,
@RequestParam(required = false) Long idParent,
                        @AuthenticationPrincipal User user) {
        Item itemToSave = new Item().setName(file.getOriginalFilename())
            .setLastModified(new Date().getTime())
            .setDirectory(false)
            .setSize(file.getSize());
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        System.out.println("Upload");
        if (idParent == null) {
           System.out.println("With No Parent");
                        if (ItemOperations.getInstance().uploadItem(file,
ItemOperations.DEFAULT_PATH + "/" + file.getOriginalFilename())) {
                itemToSave.setPath(ItemOperations.DEFAULT_PATH + "/" +
file.getOriginalFilename());
                 Map<User, ItemPermission> userItemPermissionMap = new
HashMap<>();
            userItemPermissionMap.put(currentUser, ItemPermission.OWNER);
            itemToSave.setUserItemPermissionMap(userItemPermissionMap);
            itemService.create(itemToSave);
            return new ResponseEntity<>("Uploaded", HttpStatus.ACCEPTED);
                    } else return new ResponseEntity<>("Not Uploaded",
HttpStatus.BAD_REQUEST);
        }
```

```
            System.out.println("With Parrent");
            Item item = itemService.findById(idParent).get();
            if (!ItemPermission.OWNER.equals(currentUser.getItems().get(item))) {
                                return new ResponseEntity<>("You can't upload",
HttpStatus.BAD_REQUEST);
            }
        //      itemToSave.setParent(item);
                if (ItemOperations.getInstance().uploadItem(file, item.getPath() + "/" +
file.getOriginalFilename())) {
                                itemToSave.setParent(item).setPath(item.getPath() + "/" +
file.getOriginalFilename());
                Map<User, ItemPermission> parentUserPermissions = new HashMap<>();
                parentUserPermissions.putAll(item.getUserItemPermissionMap());
                itemToSave.setUserItemPermissionMap(parentUserPermissions);
                itemService.create(itemToSave);
                return new ResponseEntity<>("Uploaded", HttpStatus.ACCEPTED);
                            } else return new ResponseEntity<>("Not Uploaded",
HttpStatus.BAD_REQUEST);


        }

        @GetMapping(value = "/download/{id}")
        @ResponseBody
                public ResponseEntity downloadItem(@PathVariable Long id,
@AuthenticationPrincipal User user) {
            Item item = itemService.findById(id).get();
            User currentUser = userService.findUserByEmail(user.getEmail()).get();
            if (item.getUserItemPermissionMap().containsKey(currentUser)) {
                return ItemOperations.getInstance().downloadItem(item.getPath());
            }

                                return new ResponseEntity<>("Can't download",
HttpStatus.BAD_REQUEST);
        }

        @PostMapping(value = "/createItem")
```

```
    @ResponseBody
    public ResponseEntity<String> createItem(@RequestParam(required = false)
Long idParent, String name, @AuthenticationPrincipal User user) {


        Item item = new Item().setDirectory(true)
//                .setParent(itemService.findById(idParent).orElse(null))
                .setParent(idParent == null ? null : itemService.findById(idParent).get())
                .setLastModified(new Date().getTime())
                .setName(name);
        User currentUser = userService.findUserByEmail(user.getEmail()).get();
        if (itemService.findByName(name) != null) {
                        new ResponseEntity<>("Such name is occupied ",
HttpStatus.BAD_REQUEST);
        }
        if (idParent != null) {
            Item itemParent = itemService.findById(idParent).get();
                                                                        if
(!ItemPermission.OWNER.equals(currentUser.getItems().get(itemParent))) {
                            return new ResponseEntity<>("Tou can't create",
HttpStatus.BAD_REQUEST);
            }
            item.setPath(itemParent.getPath() + "/" + name);
            Map<User, ItemPermission> parentUserPermissions = new HashMap<>();
            parentUserPermissions.putAll(itemParent.getUserItemPermissionMap());
            item.setUserItemPermissionMap(parentUserPermissions);
            ItemOperations.getInstance().createItem(itemParent.getPath() + "/" + name);
            item.setParent(itemParent);
            itemService.create(item);
                        return new ResponseEntity<>("Created successfully",
HttpStatus.ACCEPTED);
        }
        Map<User, ItemPermission> userItemPermissionMap = new HashMap<>();
        userItemPermissionMap.put(currentUser, ItemPermission.OWNER);
        item.setUserItemPermissionMap(userItemPermissionMap);
        item.setPath(ItemOperations.DEFAULT_PATH + "/" + name);
        itemService.create(item);
```

```
            ItemOperations.getInstance().createItem(ItemOperations.DEFAULT_PATH +
"/" + name);
                        return new ResponseEntity<>("Created successfully",
HttpStatus.ACCEPTED);
        }

        @DeleteMapping(value = "/deleteItem/{id}")
        @ResponseBody
            public ResponseEntity<String> deleteItem(@PathVariable Long id,
@AuthenticationPrincipal User user) {
            Item item = itemService.findById(id).get();
            User currentUser = userService.findUserByEmail(user.getEmail()).get();
            if (!ItemPermission.OWNER.equals(currentUser.getItems().get(item))) {
                        return new ResponseEntity<>("You can't delete",
HttpStatus.BAD_REQUEST);
            }
            itemService.delete(id);
            ItemOperations.getInstance().deleteItem(item.getPath());

                        return new ResponseEntity<>("Deleted successfully",
HttpStatus.ACCEPTED);
        }

        @PutMapping(value = "/share/{id}")
        @ResponseBody
         public ResponseEntity<String> shareItem(@PathVariable Long id, String email,
ItemPermission itemPermission,
                                    @AuthenticationPrincipal User user) {
            if (!userService.findUserByEmail(email).isPresent()) {
                        return new ResponseEntity<>("Such user doesn't exists",
HttpStatus.BAD_REQUEST);
            }
            User userToShare = userService.findUserByEmail(email).get();
            User currentUser = userService.findUserByEmail(user.getEmail()).get();
            Item itemToShare = itemService.findById(id).get();
            List<Item> itemList = itemService.getChildTree(itemToShare);
```

```
            if (!ItemPermission.OWNER.equals(currentUser.getItems().get(itemToShare)))
{
                                return new ResponseEntity<>("You cant share this",
HttpStatus.BAD_REQUEST);
            }
            for (Item item : itemList) {

               item.getUserItemPermissionMap().put(userToShare, itemPermission);

            }
            itemToShare.getUserItemPermissionMap().put(userToShare, itemPermission);
            itemService.create(itemToShare);
            return new ResponseEntity<>("Item Shared", HttpStatus.ACCEPTED);
        }

        @PutMapping(value = "/share_group/{id}")
        @ResponseBody
          public ResponseEntity<String> shareItemWithGroup(@PathVariable Long id,
Long idOfGroup, ItemPermission itemPermission,
                                              @AuthenticationPrincipal User user) {
            if (!groupService.findGroupById(idOfGroup).isPresent()) {
                            return new ResponseEntity<>("Such group doesn't exists",
HttpStatus.BAD_REQUEST);
            }

            Group groupToShare = groupService.findGroupById(idOfGroup).get();
            Set<User> usersToShare = groupToShare.getUsersWithPermissions().keySet();
            usersToShare.removeIf(user1 -> user1.getEmail().equals(user.getEmail()));
              usersToShare.forEach(userToShare -> shareItem(id, userToShare.getEmail(),
itemPermission, user));
            return new ResponseEntity<>("Item Shared", HttpStatus.ACCEPTED);
        }

        @GetMapping(value = "/users/{id}")
        @ResponseBody
              public List<UserWithItemPermission> users(@PathVariable Long id,
@AuthenticationPrincipal User user) {
```

```
        Optional<Item> item = itemService.findById(id);
        if (!item.isPresent()) {
            return new ArrayList<>();
        }
        user = userService.findUserByEmail(user.getEmail()).get();
                                    if (!user.getItems().containsKey(item.get()) ||
!ItemPermission.OWNER.equals(user.getItems().get(item.get()))) {
            return new ArrayList<>();
        }

        List<UserWithItemPermission> userWithPermissions = new ArrayList<>();
                    item.get().getUserItemPermissionMap().forEach((key, value) ->
userWithPermissions.add(new UserWithItemPermission(key, value)));
        return userWithPermissions;
    }

    @DeleteMapping(value = "/removeAccess/{id}")
    @ResponseBody
            public ResponseEntity removePermission(@PathVariable Long id,
@RequestParam String email, @AuthenticationPrincipal User currentUser) {
                                    if (!itemService.findById(id).isPresent() ||
!userService.findUserByEmail(email).isPresent()) {
            return new ResponseEntity(HttpStatus.BAD_REQUEST);
        }
        currentUser = userService.findUserByEmail(currentUser.getEmail()).get();


        User user = userService.findUserByEmail(email).get();
        Item itemToRevertAccess = itemService.findById(id).get();
        List<Item> itemList = itemService.getChildTree(itemToRevertAccess);
        for (Item item : itemList) {
            item.getUserItemPermissionMap().remove(user);
            itemService.create(item);
        }
        itemToRevertAccess.getUserItemPermissionMap().remove(user);
```

```
        itemService.create(itemToRevertAccess);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
package ua.kpi.tef.entity;

import lombok.AllArgsConstructor;

import javax.persistence.*;
import java.io.File;
import java.util.Map;

@AllArgsConstructor
@Entity
@Table(name = "item")
public class Item {
    public Item(File file) {
        path = file.getPath();
        isDirectory = file.isDirectory();
        name = file.getName();
        size = file.length();
        lastModified = file.lastModified();
    }

    public Item() {
    }

    @ElementCollection
    @CollectionTable(name = "user_to_item",
            joinColumns = @JoinColumn(name = "id"))
    @MapKeyJoinColumn(name = "email")
    @Column(name = "item_permission")
    private Map<User, ItemPermission> userItemPermissionMap;



    @GeneratedValue
```

```
@Id
private Long itemId;

@ManyToOne
private Item parent;

@Column(name = "path")
private String path;

@Column(name = "isDirectory")
private boolean isDirectory;

@Column(name = "name")
private String name;

@Column(name = "lastModified")
private Long lastModified;

@Column(name = "size")
private Long size;


public Map<User, ItemPermission> getUserItemPermissionMap() {
   return userItemPermissionMap;
}

        public void setUserItemPermissionMap(Map<User, ItemPermission>
userItemPermissionMap) {
   this.userItemPermissionMap = userItemPermissionMap;
}


public Long getItemId() {
   return itemId;
}

public void setItemId(Long itemId) {
```

```
    this.itemId = itemId;
}

public Item getParent() {
    return parent;
}

public Item setParent(Item parent) {
    this.parent = parent;
    return this;
}

public String getPath() {
    return path;
}

public Item setPath(String path) {
    this.path = path;
    return this;
}

public boolean isDirectory() {
    return isDirectory;
}

public Item setDirectory(boolean directory) {
    isDirectory = directory;
    return this;
}

public String getName() {
    return name;
}

public Item setName(String name) {
    this.name = name;
    return this;
```

```
    }

    public Long getLastModified() {
        return lastModified;
    }

    public Item setLastModified(Long lastModified) {
        this.lastModified = lastModified;
        return this;
    }

    public Long getSize() {
        return size;
    }

    public Item setSize(Long size) {
        this.size = size;
        return this;
    }
}
```

ДОДАТОК В

Розробка програмних засобів для авторизованого доступу до кафедрального хмарного сховища.

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51157_19Б 13-1

Аркушів 8

Київ 2019

АНОТАЦІЯ

Розділ містить опис частини, що відповідає за контрольований доступ до кафедрального хмарного сховища. Цей модуль є дуже вагомим, бо відповідає за безпеку даних, що знаходяться у даному сховищі. Основною функцією модулю є отримання даних авторизації від користувача, аналіз цих даних та пропуск до хмарного сховища, якщо дані відповідають вимогам. Модуль написано на мові Java з використанням фреймворку Spring та технології OAuth2.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку розглядається один із основних модулів системи — модуль контрольованого доступу до кафедрального хмарного сховища з кодом УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51157_19Б 12-1, що міститься у файлі SecurityConfiguration.java. Даний модуль було реалізовано за допомогою технології OAuth2. Дана технологія розроблена компанією Google для полегшення аутентифікації користувачів веб-ресурсів. Щоб пройти авторизацію, користувачу лише потрібно пройти авторизацію у своєму акаунті гугл та переконатися, що його було запрошено до кафедрального хмарного сховища.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням модулю контрольованого доступу є захист даних, що знаходяться у кафедральному хмарному сховищі. Було використано саме технологію OAuth2, бо це значно полегшує авторизацію з боку користувача системи та надає системі велику кількість відомостей про самого користувача. Дана технологія є універсальною для всіх веб-додатків, що дає можливість використовувати дану технологію при масштабуванні системи та при розробці нової.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Головним завданням модуля є організація контрольованого доступу до кафедрального хмарного сховища. У випадку коли система потребує авторизації користувача, виконується переадресація на вікно авторизації. У даному вікні авторизації необхідно ввести свої дані до акаунту Google. Це потребується зробити лише раз, при подальшому використанні сервісу повторне введення даних не потрібне до закінчення сесії користування системою. Відповідальний сервіс Google повертає системі дані про користувача, система аналізує ці дані, та якшо вони відповідають вимогам, то надає доступ до хмарного сховища.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Даний модуль було розроблено у середовищі розробки IntelliJ IDEA Ultimate, що надає велику підтримку в написанні коду розробником. Дане середовище має великий набір допоміжних функцій та бібліотек, які полегшують написання коду. Попередня конфігурація технологія OAuth2 виконувалась на відповідному сервісі Google, а підтримка цієї технології системою була реалізована за допомогою фреймворку Spring.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Даний програмний модуль реалізовано як пакет декількох класів, що виконуються автоматично. Для використання цього модулю необхідно щоб система запросила у користувача дані його авторизації. Користувач може самостійно скористатись цим модулем, натиснувши на кнопку “Увійти”.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для модуля є дані, які предоставляє Google після успішної авторизації користувача.

Вихідними даними модулю є дані, що ідентифікують користувача і також безпосередній доступ до кафедрального хмарного сховища.